A User Guide to the PROCESS Systems Code

P. J. Knight

Culham Centre for Fusion Energy
Culham Science Centre, Abingdon, Oxon, OX14 3DB, UK

Contents

1 Introduction **9**

1.1 Rationale . . . 9

1.2 History . . . 10

1.3 Layout of the User Guide . . . 10

2 Program Overview — The Fundamental Concepts **11**

2.1 Equation Solvers . . . 11

2.1.1 Non-optimisation mode . . . 11

2.1.2 Optimisation mode . . . 11

2.1.3 Scans . . . 13

2.2 The Variable Descriptor File . . . 13

2.3 Input Parameters . . . 15

2.4 Constraint Equations . . . 15

2.4.1 Consistency equations . . . 15

2.4.2 Limit equations . . . 15

2.5 Iteration Variables . . . 16

2.6 Figures of Merit . . . 18

2.7 Scanning Variables . . . 18

2.8 Code Structure . . . 18

2.8.1 Numerics modules . . . 21

2.8.2 Physics modules . . . 21

2.8.3 Engineering modules . . . 21

2.8.4 Costing module . . . 23

2.8.5 Other modules . . . 23

3 Physics, Engineering and Other Models **25**

3.1 Tokamak Power Plant . . . 25

3.1.1 Radial and vertical build . . . 25

3.1.2 Plasma physics models . . . 29
3.1.3 First wall . . . 38
3.1.4 Divertor . . . 39
3.1.5 Blanket . . . 39
3.1.6 Shield . . . 42
3.1.7 Toroidal field coils . . . 43
3.1.8 Poloidal field coils . . . 45
3.1.9 Ohmic heating coil . . . 46
3.1.10 Auxiliary power systems: heating and current drive . . . 48
3.1.11 Structural components . . . 51
3.1.12 Cryostat and vacuum system . . . 51
3.1.13 Power conversion and heat dissipation systems . . . 51
3.1.14 Buildings . . . 54
3.2 Spherical Tokamak Model . . . 55
3.2.1 Spherical tokamak switches . . . 55
3.3 Pulsed Plant Operation . . . 56
3.3.1 Thermal cycling package . . . 56
3.3.2 First wall coolant temperature rise limit . . . 56
3.3.3 First wall peak temperature limit . . . 56
3.3.4 Start-up power requirements . . . 56
3.3.5 Plasma current ramp-up time . . . 57
3.3.6 Burn time . . . 57
3.3.7 Thermal storage . . . 57
3.4 Hydrogen Production Facility . . . 57
3.5 Stellarator Model . . . 58
3.5.1 Stellarator physics . . . 58
3.5.2 Machine configuration . . . 62
3.6 Reversed Field Pinch Model . . . 63
3.6.1 RFP physics . . . 63
3.6.2 TF coils . . . 64
3.6.3 Ohmic heating coils . . . 65
3.6.4 EF coils . . . 65
3.6.5 Divertor . . . 65
3.6.6 Code modifications . . . 65
3.7 Inertial Fusion Energy Model . . . 65

3.8 Safety and Environment Models . . . 66
3.8.1 Neutronics . . . 66
3.8.2 Activation and inventory information . . . 67
3.9 Cost Models . . . 67
3.9.1 Cost options . . . 67
3.10 Other Switches and Models . . . 68
3.10.1 Diagnostic output . . . 68
3.10.2 Code parameters affecting other models . . . 68

4 Execution of the Code 69
4.1 The Input File . . . 69
4.1.1 Tokamak, stellarator, RFP or IFE? . . . 69
4.1.2 File structure . . . 69
4.1.3 Format rules . . . 70
4.2 The Output File . . . 71
4.3 Running the Code . . . 71
4.3.1 Environment set-up . . . 72
4.3.2 Non-optimisation mode . . . 72
4.3.3 Optimisation mode . . . 74
4.4 Problem Solving . . . 75
4.4.1 Error handling . . . 75
4.4.2 General problems . . . 76
4.4.3 Optimisation problems . . . 76
4.4.4 Unfeasible results . . . 76
4.4.5 Hints . . . 78

5 Inclusion of New Models, Additional Variables and Equations 79
5.1 Input Parameters . . . 79
5.2 Iteration Variables . . . 79
5.3 Other Global Variables . . . 80
5.4 Constraint Equations . . . 80
5.5 Figures of Merit . . . 81
5.6 Scanning Variables . . . 81
5.7 Submission of New Models . . . 82

6 Utility Programs 83
6.1 Executables . . . 83

6.1.1 write_new_in_dat.py . . . 83
6.1.2 write_constraints.py . . . 84
6.1.3 run_process.py . . . 84
6.1.4 build_index.py . . . 85
6.1.5 make_plot_dat.py . . . 86
6.1.6 plot_proc.py . . . 87
6.1.7 plot_sweep.py . . . 87
6.1.8 plot_mfile_sweep.py . . . 88
6.1.9 diagnose_process.py . . . 88
6.1.10 2D_scan.py . . . 89
6.1.11 a_to_b.py . . . 90
6.1.12 create_dicts.py . . . 91
6.2 Libraries . . . 91
6.2.1 in_dat.py . . . 91
6.2.2 mfile.py . . . 92
6.2.3 process_funcs.py . . . 93
6.2.4 process_config.py . . . 94
6.2.5 process_dicts.py . . . 96
6.2.6 a_to_b_config.py . . . 96
6.2.7 proc_plot_func.py . . . 97
6.3 User Interface . . . 97
6.3.1 Launching . . . 97
6.3.2 Editing variables . . . 98
6.3.3 Opening and saving files . . . 98
6.3.4 Searching . . . 98

7 Code Management Tools **99**
7.1 The Makefile . . . 99
7.1.1 Compilation . . . 99
7.1.2 Archiving utilities . . . 100
7.1.3 Code documentation . . . 100
7.2 Automatic Documentation . . . 100

8 Acknowledgements & Bibliography **101**

A The Optimisation Solver Explained **106**
A.1 The General Nonlinear Programming Problem . . . 106

A.2 The Lagrange Method 108
A.3 Sequential Quadratic Programming (SQP) 109
A.4 `VMCON` 109
A.5 The Quadratic Subproblem (QSP) 109
A.6 The Line Search 112
A.7 The Broyden-Fletcher-Goldfarb-Shanno (BFGS) quasi-Newton update 113
A.8 Symbols 114

B Non-optimisation Input File **115**

C Optimisation Input File **118**

D Source Code Documentation **121**

List of Figures

2.1 Flow diagram of PROCESS in non-optimisation mode 12
2.2 Flow diagram of PROCESS in optimisation mode 14

3.1 Machine build for D-shaped major components 26
3.2 Machine build for elliptical-shaped major components 27
3.3 Machine build for a single-null device 28
3.4 Power balance within the core plasma 36
3.5 Schematic diagram of the cross-section of a superconducting TF coil inner leg 44
3.6 Coil and plasma current waveforms 47
3.7 Schematic diagram of the neutral beam access geometry 50
3.8 Power flow within the fusion power plant core 52
3.9 Power flow outside the fusion power plant core 54
3.10 HELIAS 5-B Stellarator Power Plant Design 59
3.11 Stellarator island divertor configuration 62

A.1 Illustration of Lagrange multiplier method 107
A.2 Flowchart of the VMCON optimisation solver 110

List of Tables

2.1 List of constraint equations 17
2.2 List of iteration variables 1 to 50 19
2.3 List of iteration variables 51 to 102 20
2.4 List of figures of merit 21
2.5 List of scanning variables 22
2.6 Summary of numerics modules 22
2.7 Summary of physics modules 23
2.8 Summary of engineering modules 23
2.9 Summary of other modules 24

3.1 List of available energy confinement scaling laws 35
3.2 List of available L-H power threshold scalings 38
3.3 Summary of blanket material scenarios 42
3.4 `PROCESS` switches for spherical tokamaks 56
3.5 Variables used in the hydrogen plant model 58

A.1 Summary of the description and meaning of the `VMCON` return parameters `ifail`. . . . 111

Chapter 1

Introduction

1.1 Rationale

During the course of studies into a proposed fusion power plant, there may be times when questions of the following type arise:

> Are the machine's physics and engineering parameters consistent with one another?
>
> Which machine of a given size and shape produces the cheapest electricity?
>
> What is the effect of a more optimistic limit on the maximum plasma density on the amount of auxiliary power required?

Questions such as these are extremely difficult to answer, since the large number of parameters involved are highly dependent on one another. Fortunately, computer programs have been written to address these issues, and `PROCESS` is one of them.

Suppose that an outline power plant design calls for a machine with a given size and shape, which will produce a certain net electric power. There may be a vast number of different conceptual machines that satisfy the problem as stated so far, and `PROCESS` can be used in "non-optimisation" mode to find one of these whose physics and engineering parameters are self-consistent. However, the machine found by `PROCESS` in this manner may not be possible to build in practice — the coils may be overstressed, for instance, or the plasma pressure may exceed the maximum possible value. `PROCESS` contains a large number of constraints to prevent the code from finding a machine with such problems, and running the code in so-called "optimisation" mode forces these constraints to be met. The number of possible conceptual machines is thus considerably reduced, and optimisation of the parameters with respect to (say) the cost of electricity will reduce this number to a minimum (possibly one).

Formally then, `PROCESS` is a systems code that calculates in a self-consistent manner the parameters of a fusion power plant with a specified performance, ensuring that its operating limits are not violated, and with the option to optimise a given function of these parameters.

It would not be fair to call `PROCESS` a fusion power plant *design* code, as this implies that a great deal of complexity would need to be present in each and every model describing one of the component systems. Such complexity is, however, incompatible with the code's iterative approach to solving the optimisation problem, since this requires repeated evaluation of the same (large number of) expressions. This is not to say that the models employed by the code are oversimplified — in general they represent good numerical estimates of present theoretical understanding, or are fits to experimental data. `PROCESS` provides a useful overall description of how a conceptual and feasible power plant may look.

1.2 History

PROCESS is derived from several earlier systems codes, but is largely based on the TETRA (Tokamak Engineering Test Reactor Analysis) code [1] and its descendant STORAC (Spherical TOrus Reactor Analysis Code) [2], which includes routines relevant to the spherical tokamak class of machines. These codes, and much of the original version of PROCESS itself, were written by personnel at Oak Ridge National Laboratory in Tennessee, USA, with contributions from a number of other laboratories in the USA. In addition, many of the mathematical routines have been taken from a number of different well-established source libraries.

Since the code is descended from such a wide range of sources, its structure was initially not ideal from the programmer's viewpoint. Non-standard practices and inconsistent layout within the code led to difficulties in modifying, interpreting and indeed running the code. A great deal of effort was therefore expended at Culham on the code's arrival from ORNL in the early 1990s to improve this situation, with the code being given a complete but careful upgrade, routine by routine. For many years this Fortran 77 code was used for systems code studies of various power plant scenarios, and was modified from time-to-time by the addition of new and/or improved models, including machines based on the stellerator, reversed field pinch and inertial confinement concepts.

In 2012, the code structure was revised again to allow it to benefit from modern software practices, and the whole program was upgraded to Fortran 90/95. At the same time a number of useful code management utilities were added.

As with all active research codes, PROCESS will continue to be developed into the future. This User Guide is updated in parallel with the Fortran source code itself to ensure that the documentation remains consistent with the latest version of the code. It is to be hoped that it will be of assistance to all users of PROCESS, whether they are planning to modify or run the code, or are simply trying to understand what the code aims to achieve.

1.3 Layout of the User Guide

The User Guide is divided into a small number of logically separate chapters, each one of which provides specific information on a given topic. It depends on the user's motive for referring to the manual as to which chapter will be the most useful, although hopefully the style and structure adopted will allow one to browse through without difficulty.

Chapter 2 provides an overview of the program, and outlines the numerical and programming concepts involved. Chapter 3 describes the physics, engineering and economic models that are used within the code, and lists the switches available allowing the user to customise the models' details to achieve the desired simulation. Chapter 4 describes how to run the program from scratch, and provides a number of hints and suggestions for the user to bear in mind to help the code find a feasible machine. Chapter 5 shows how to modify the code in specific ways, for example how to add extra constraints and variables to the code. A useful set of utility programs is introduced in Chapter 6, and some code management tools are described in Chapter 7. Finally, the Appendices give detailed information about the optimisation method, example input files for PROCESS in non-optimisation and optimisation modes, and lists of references that provide information about the code status, its location, and other details relating to the implementation of PROCESS to date.

Chapter 2

Program Overview — The Fundamental Concepts

Fusion power plants are complex systems consisting of many non-linear interactions. One method that can be used to model this kind of system is to iterate a number of free parameters (the so-called *iteration variables* — see Section 2.5) in a controlled way so as to find a self-consistent set of device parameters that satisfy all of the system's *constraint equations* — see Section 2.4. PROCESS is organised in a standard equation solver format to enable this task to be performed efficiently. The physics and engineering routines together serve as a *function evaluator*, providing the information used in the solution of the constraints. The numerical modules present in PROCESS perform the iteration required, and also incorporate the option to maximise or minimise a given *figure of merit* – see Section 2.6.

2.1 Equation Solvers

PROCESS contains two non-linear equation solver packages, which reflect the two major modes of operation available. Each of these has its own uses, as is now discussed.

2.1.1 Non-optimisation mode

The first of the two equation solvers present in PROCESS is the non-optimisation package HYBRD [3, 4]. Formally, HYBRD finds a zero of a system of N non-linear functions in N variables. This means simply that N variables (power plant parameters) are iterated by PROCESS in such a way as to solve a set of N equations (physics or engineering laws), i.e. a set of self-consistent power plant parameters is found. This is useful for performing benchmark comparisons, when the device size is kept fixed, and one only wishes to find calculated stresses, beta values, fusion powers, etc. A flow diagram of PROCESS in non-optimisation mode is shown in Figure 2.1.

2.1.2 Optimisation mode

Sometimes one wants to find an optimal machine that is both consistent with the physics and engineering constraints but also minimises or maximises a certain *figure of merit*. This requires running PROCESS in optimisation mode. For these applications PROCESS uses the routine VMCON [5] based on a variable metric method for constrained optimisation by Powell [6]. It finds a stationary point of an objective function/figure of merit consistent with a set of equality and inequality constraints

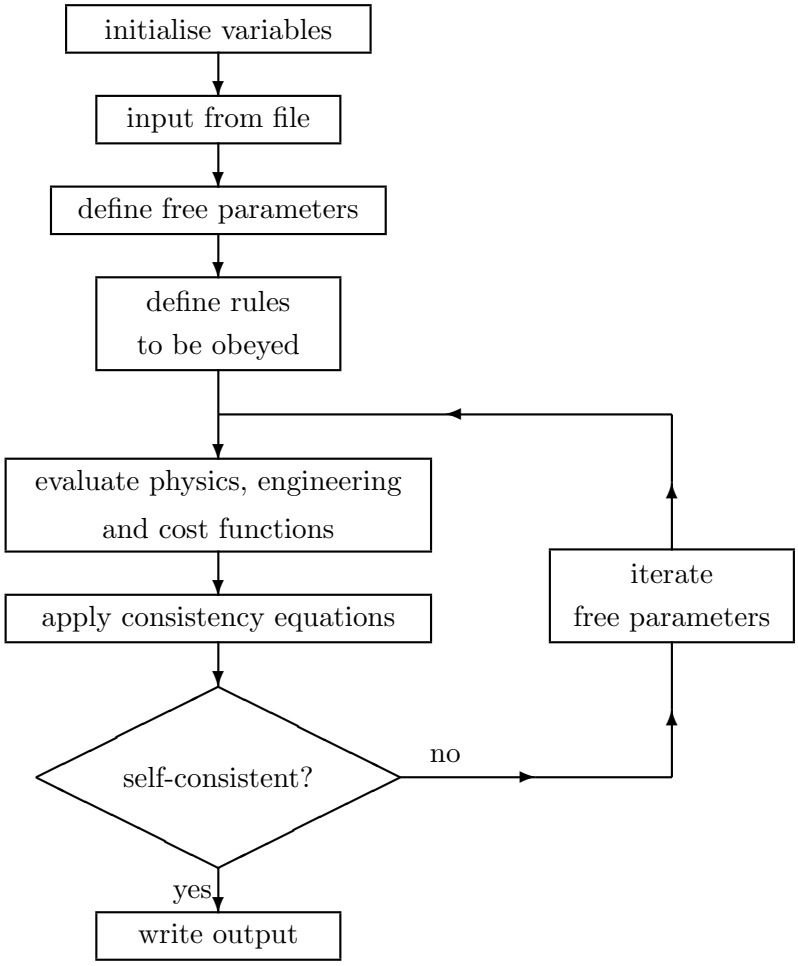

Figure 2.1: *Flow diagram of* `PROCESS` *in non-optimisation mode.*

(c.f. section A.1). It is designed to solve the necessary but not sufficient conditions of a constrained optimum, hence, the solution does not have to be a global optimum (c.f. A.2).

The detailed algorithm is explained in Appendix A and is based on the Lagrange method (c.f. section A.2) that combines both the objective function and the constraints using Lagrange multipliers. It applies a sequential quadratic programming approach (c.f. section A.3) in which a series of subproblems is solved that constitute a local quadratic expansion of the Lagrangian (c.f. section A.5). The solution of the quadratic subproblem is the direction of a line search along which a one dimensional optimisation is performed (c.f. section A.6). This line search was introduced by Powell to assure convergence from bad starting parameters. Within `PROCESS` we have modified the line search of the original `VMCON` routine to assure convergence, even for slightly inconsistent input functions.

The algorithm uses a quasi-Newtonian approach which only requires continuous first derivatives of these functions. While the first derivatives are evaluated using a finite difference approximation, the second derivatives are estimated using a variant of the Broyden-Fletcher-Goldfarb-Shanno update (c.f. section A.7).

The convergence criterium of the solver as well as the detailed interpretation of the various error codes are explained in section A.4. A flow diagram of `PROCESS` in optimisation mode is shown in Figure 2.2.

2.1.3 Scans

It is often useful to be able to scan through a range of values of a given parameter to see what effect this has on the machine as a whole. Sensitivity studies of this kind can be achieved very easily using `PROCESS`. Scans are carried out in optimisation mode, whereby the code performs initially a run using the parameters specified in the input file, and then a series of runs using the parameters produced at the end of the previous iteration. The value of the quantity being scanned is specified at every stage — see Section 2.7. This method ensures that a smooth variation in the machine parameters is achieved.

2.2 The Variable Descriptor File

The variable descriptor file `vardes.html` is an invaluable resource for the user of `PROCESS`. It acts as a dictionary / reference manual for the code's variables, and contains the following information about each:

- name
- dimensions (of arrays)
- default value(s) of those variables that are not initially derived from a combination of other values. The default values are mostly set in the modules contained within source file `global_variables.f90`.
- description, including physical units if relevant
- for switches/flags, the meanings of all allowed values
- iteration variable number, if relevant
- corresponding constraint equation, if relevant

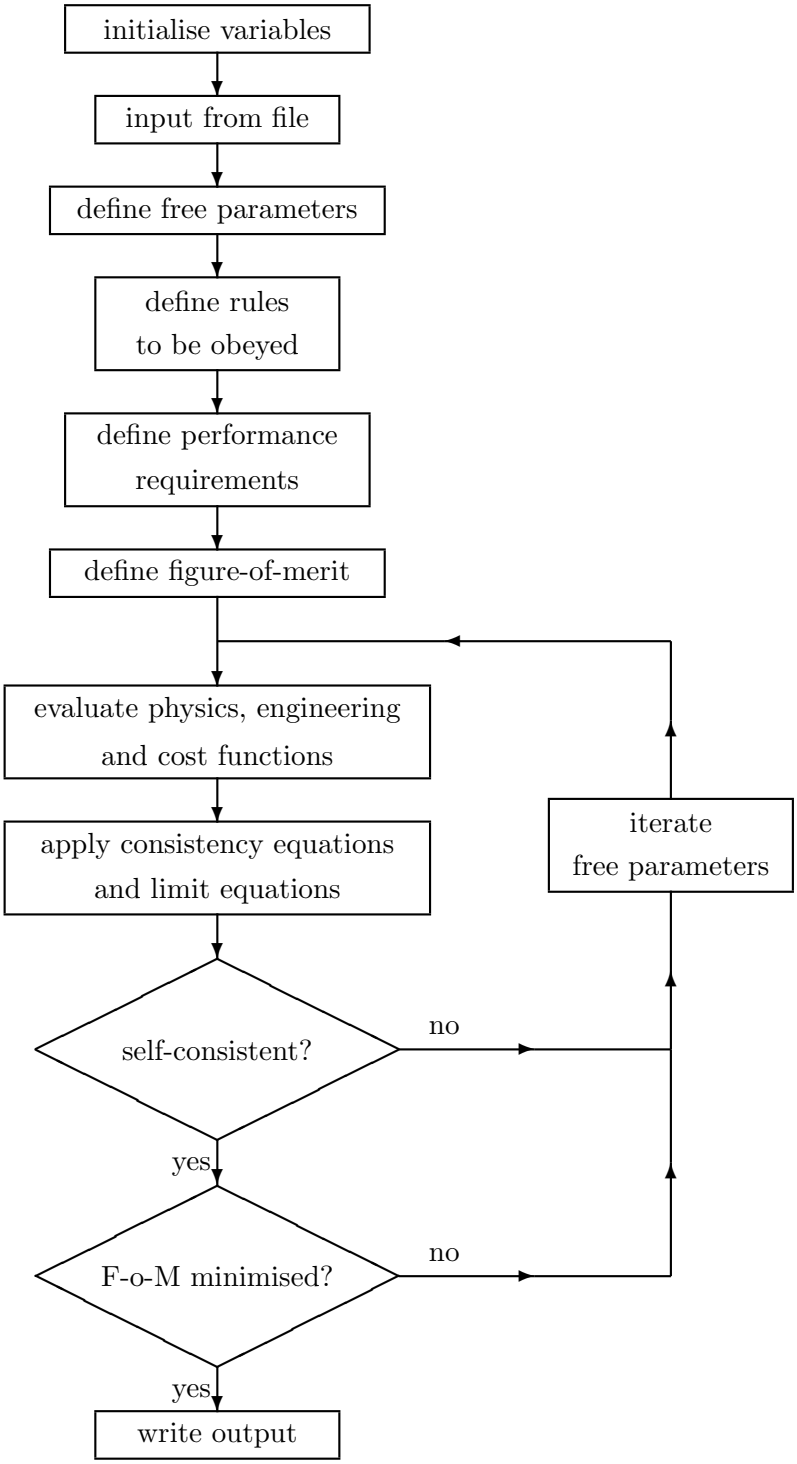

Figure 2.2: *Conceptual flow diagram of* `PROCESS` *in optimisation mode. Please note that this is a simplistic interpretation of the actual sequence of operations, to outline the difference between non-optimisation and optimisation modes. For a more detailed (and correct!)* `VMCON` *flow diagram, please see Figure A.2.*

In addition, global code parameters are labelled FIX. These can only be changed by editing the relevant source file, but this should not be carried out unless it is absolutely necessary.

All the variables that are shown with a default value are available to be changed by the user using the input file (Section 4.1), except for those which are labelled FIX. Variables not shown with a default value are calculated by the code from a combination of other parameters, and so it would be meaningless to initialise them. Obviously, these variables cannot be changed using the input file.

The file is generated from specially-formatted comment lines within the source code (see Section 7.2 for more details). Therefore, it is exceedingly important to keep these comment lines relevant and in sync with the variables they describe.

2.3 Input Parameters

Input parameters make up a large proportion of the variables listed in the variable descriptor file. They comprise all those variables that, once set in the initialisation routine or redefined in the input file, do not change throughout a PROCESS run. In fact, only those variables defined as iteration variables (Section 2.5) can change during the course of a run.

2.4 Constraint Equations

Any computer program naturally contains myriads of equations. The built-in equation solvers within PROCESS act on a special class, known as *constraint equations*, all of which are formulated in routine CONSTRAINTS in source file constraint_equations.f90. Table 2.1 summarises the constraint equations available in PROCESS. These can be split into two types — (1) consistency equations, that enforce consistency between the physics and engineering parameters, and (2) limit equations, that enforce various parameters to lie within their allowed limits. The neqns constraint equations that the user chooses for a given run are activated by including the equation numbers in the first neqns elements of array icc.

2.4.1 Consistency equations

Consistency equations are usually *equalities* that ensure that the machine produced by PROCESS is self-consistent. This means, therefore, that many of these constraint equations should *always* be used, namely equations 1, 2, 10 and 11 (see Table 2.1). Equation 7 should also be activated if neutral beam injection is used. The other consistency equations can be activated if required.

A typical consistency equation ensures that two functions g and h are equal:

$$\begin{aligned} g(x, y, z, \ldots) &= h(x, y, z, \ldots) \\ c_i &= 1 - \frac{g}{h} \end{aligned}$$

The equation solvers VMCON and HYBRD need the constraint equations c_i to be given in the form shown, since they adjust the iteration variables so as to obtain $c_i = 0$, thereby ensuring that $g = h$.

2.4.2 Limit equations

The limit equations are usually *inequalities* that ensure that various physics or engineering limits are not exceeded. Each of these equations has an associated *f-value*, which allow them to be coded as

equalities. The f-values are used as follows.

In optimisation mode, all iteration variables have prescribed lower and upper bounds. In general, limit equations have the form

$$\textit{calculated quantity} = f \times \textit{maximum allowable value}$$

where f is the f-value. If f has a lower bound of zero and an upper bound of one, then the limit equation does indeed constrain the calculated quantity to lie between zero and its maximum allowable value, as required.

As with the consistency equations, the general form of the limit equations is

$$c_i = 1 - f.\frac{h_{\max}}{h}$$

where $h_{\max}$ is the maximum allowed value of the quantity h.

Sometimes, the limit equation and f-value are used to ensure that quantity h is *larger* than its *minimum* value $h_{\min}$. In this case, $0 \leq f \leq 1$ (as before), but the equation takes the form

$$c_i = 1 - f.\frac{h}{h_{\min}}$$

By fixing the f-value (i.e. not including it in the `ixc` array), the limit equations can be used as equality constraints. For example, to set the net electric power to a certain value, the following should be carried out:

1. Activate constraint equation 16 by including it in the first `neqns` elements of array `icc`
2. Set `fpnetel = 1.0D0`
3. Ensure that `fpnetel` (iteration variable no. 25) *DOES NOT* appear in array `ixc`
4. Set `pnetelin` to the required net electric power.

Limit equations are not restricted to optimisation mode. In non-optimisation mode, the iteration variables are not bounded, but the f-values can still be used to provide information about how calculated values compare with limiting values, without having to change the characteristics of the device being benchmarked to find a solution.

It is for this reason that all the constraint equations used in `PROCESS` are formulated as equalities, despite the fact that equation solver `VMCON` can solve for inequalities as well. The use of f-values precludes this need, and allows the non-optimising equation solver HYBRD to use the same constraint equations.

2.5 Iteration Variables

It is necessary to calculate numerical derivatives during the solution of the constraint equations. The iteration variables are the parameters that the equation solvers use for this purpose — all the other code variables (input parameters — see above) remain fixed at their initial value. Successive calls are made to the physics and engineering routines, with slightly different values for the iteration variables on each call, and the equation solver determines the effect on the output due to these small changes

icc no.	description	type	corresponding ixc variables
1	plasma beta consistency	C	**5**
2	global power balance	C	**10**,1,2,3,4,6,11
3	ion power balance	C	**10**,1,2,3,4,6,11
4	electron power balance	C	**10**,1,2,3,4,6,11
5	density upper limit	L	**9**,1,2,3,4,5,6
6	epsilon-beta poloidal upper limit	L	**8**,1,2,3,4,6
7	beam ion density (NBI)	C	**7**
8	neutron wall load upper limit	L	**14**,1,2,3,4,6
9	fusion power upper limit	L	**26**,1,2,3,4,6
10	toroidal field 1/R consistency	C	**12**,1,2,3,13
11	radial build consistency	C	**3**,1,13,16,29,42,61
12	volt second capability lower limit	L	**15**,1,2,3
13	burn time lower limit (PULSE)	L	**21**,1,16,17,22,29,42,44,61
14	beam energy (NBI)	C	**19**,1,2,3,6
15	UNUSED		
16	net electric power lower limit	L	**25**,1,2,3
17	radiation power upper limit	L	**28**
18	divertor heat load upper limit	L	**27**
19	MVA upper limit	L	**30**
20	neutral beam tangency radius upper limit (NBI)	L	**33**,31,3,13
21	minor radius lower limit	L	**32**
22	divertor collisionality upper limit	L	**34**,43
23	UNUSED		
24	Beta upper limit (also beta limit in stellarators)	L	**36**,1,2,3,4,6,18
25	peak toroidal field upper limit	L	**35**,3,13,29
26	central solenoid current density at End of Flat-top upper limit	L	**38**,37,41,12
27	central solenoid current density at Beginning of Pulse upper limit	L	**39**,37,41,12
28	fusion gain Q lower limit	L	**45**,47,40
29	inboard radial build = specified value	C	**3**,1,13,16,29,42,61
30	injection power upper limit	L	**46**,47,11
31	TF coil case stress upper limit (SCTF)	L	**48**,56,57,58,59,60,24
32	TF coil conduit stress upper limit (SCTF)	L	**49**,56,57,58,59,60,24
33	TF coil $I_{\text{operational}}/I_{\text{critical}}$ upper limit (SCTF)	L	**50**,56,57,58,59,60,24
34	TF coil dump voltage upper limit (SCTF)	L	**51**,52,56,57,58,59,60,24
35	TF coil $J_{\text{winding pack}}/J_{\text{protection}}$ upper limit (SCTF)	L	**53**,56,57,58,59,60,24
36	TF coil temperature margin lower limit (SCTF)	L	**54**,55,56,57,58,59,60,24
37	current drive gamma upper limit	L	**40**,47
38	first wall coolant temperature rise upper limit (PULSE)	L	**62**
39	first wall peak temperature upper limit (PULSE)	L	**63**
40	injection power lower limit (PULSE)	L	**64**
41	central solenoid current ramp-up time lower limit (PULSE)	L	**66**,65
42	cycle time lower limit (PULSE)	L	**67**,65,17
43	average centrepost temperature (ST)	C	**69**,70,13
44	peak centrepost temperature upper limit (ST)	L	**68**,69,70
45	edge safety factor lower limit (ST)	L	**71**,1,2,3
46	Ip/Irod upper limit (ST)	L	**72**,2,60
47	TF coil toroidal thickness upper limit (RFP)	L	**76**,77,13,3
48	poloidal beta upper limit	L	**79**,2,3,18
49	reversal parameter < 0 (RFP)	L	**80**,78,3,1
50	IFE repetition rate upper limit (IFE)	L	**86**
51	startup volt-seconds consistency (PULSE)	C	**16**,29,3,1
52	tritium breeding ratio lower limit	L	**89**,90,91
53	peak neutron fluence on TF coil upper limit	L	**92**,93,94
54	peak TF coil nuclear heating upper limit	L	**95**,93,94
55	final He concentration in vacuum vessel upper limit	L	**96**,93,94
56	$P_{\text{separatrix}}/R_{\text{major}}$ upper limit	L	**97**,1,3
57	(OBSOLETE) TF coil outer leg toroidal thickness lower limit (SCTF)	L	**99**,29,13
58	(OBSOLETE) TF coil outer leg radial thickness limit lower (SCTF)	L	**100**,13

Table 2.1: *Summary of the constraint equations present in* PROCESS*. Consistency equations are marked C, limit equations are marked L. Some (non-exhaustive) iteration variable numbers (see Tables 2.2 and 2.3) that directly affect the associated constraint equations are given, the one listed first being the most relevant.*

to the input (see Figures 2.1 and 2.2). The `nvar` iteration variables that the user chooses for a given run are activated by including the variable numbers in the first `nvar` elements of array `ixc`. Tables 2.2 and 2.3 list the iteration variables available in `PROCESS`.

Clearly, the equation solvers need at least as many variables to iterate as there are equations to solve, i.e. `nvar` $\geq$ `neqns`. If the run is a non-optimising case, then `neqns` variables are iterated — the values of the remaining `(nvar-neqns)` variables are left alone. If the run is an optimising case, then all the active iteration variables are adjusted so as to find the minimum (or maximum) value of a parameter (the *figure of merit*) in the `nvar`-dimensional space of the problem.

All the iteration variables are constrained to lie between lower and upper bounds, stored in arrays `boundl` and `boundu`, respectively. For instance, the plasma electron density is, by default, confined to lie between the values 10^{19} m^{-3} and 10^{21} m^{-3}. Of course, it can also be constrained to lie below the *calculated* density limit, if constraint equation 5 is activated and the f-value `fdene` (iteration variable no. 9) is bounded by the values 0 and 1.

It is important to remember that iteration variables *must never be initialised to zero*. The code will not be able to adjust the variable's value if this is done, and it will stop with an error message.

2.6 Figures of Merit

In optimisation mode, `PROCESS` finds the self-consistent set of iteration variable values that maximises or minimises a certain function of them, known as the *figure of merit*. Several possible figures of merit are available, all of which are formulated in routine `FUNFOM` in source file `evaluators.f90`. Switch `minmax` is used to control which figure of merit is to be used, as summarised in Table 2.4. If the figure of merit is to be minimised, `minmax` should be positive, and if a maximised figure of merit is desired, `minmax` should be negative.

2.7 Scanning Variables

One of a number of variables can be scanned during the course of a `PROCESS` run. This option provides a method of determining the sensitivity of the results to different input assumptions. The user specifies which variable is to be scanned (see Table 2.5) and its required value at each point in the scan. The scanned variable to use is defined by the value of `nsweep`, and the chosen variable's values during the scan are set in array `sweep`.

Runs involving scans of this kind can only be performed in optimisation mode. The results from the previous scan point are used as the input to the next scan point. Routine `SCAN` in source file `scan.f90` stores many of the output quantities in a separate output file called `PLOT.DAT`, which can be read by the utility program `plot_sweep` to produce graphical output (see Chapter 6).

Scanning of derived quantities requires use of the appropriate constraint equations. For instance, if the net electric power is scanned, constraint equation 16 should be employed.

For obvious reasons, the active scanning variable must not also be an active iteration variable.

2.8 Code Structure

The structure of the majority of the code reflects to a certain extent the layout of the machine being modelled. As stated above, a large proportion of the code is simply a description of the underlying

ixc no.	variable name	description	icc eqn	lower bound	upper bound
1	aspect	plasma aspect ratio		1.100D0	10.00D0
2	bt	toroidal field on axis		0.010D0	100.0D0
3	rmajor	plasma major radius		0.100D0	10.00D0
4	te	electron temperature		5.000D0	500.0D0
5	beta	plasma beta		0.001D0	1.000D0
6	dene	electron density		1.00D19	1.00D21
7	rnbeam	hot beam density / electron density		1.00D-6	1.000D0
8	fbeta	f-value for $\epsilon.\beta_p$ limit equation	6	0.001D0	1.000D0
9	fdene	f-value for density limit equation	5	0.001D0	1.000D0
10	hfact	confinement time H-factor		0.100D0	3.000D0
11	pheat	heating power not used for current drive		0.001D0	1.000D3
12	oacdcp	overall current density in TF coil inboard leg		1.000D5	1.500D8
13	tfcth	TF coil inboard leg thickness		1.000D0	5.000D0
14	fwalld	f-value for wall load limit equation	8	0.001D0	1.000D0
15	fvs	f-value for volt second limit equation	12	0.001D0	1.000D0
16	ohcth	central solenoid thickness		0.001D0	1.000D2
17	tdwell	dwell time		0.100D0	1.000D8
18	q	edge safety factor		2.000D0	100.0D0
19	enbeam	neutral beam energy		1.000D0	1.000D6
20	tcpav	average (resistive) TF coil temperature		40.00D0	1.000D3
21	ftburn	f-value for burn time limit equation	13	0.001D0	1.000D0
22	tbrnmn	minimum burn time		0.001D0	1.000D6
23	fcoolcp	coolant fraction of resistive TF coil		0.100D0	0.500D0
24	cdtfleg	TF coil leg overall current density		1.000D4	1.000D8
25	fpnetel	f-value for net electric power limit equation	16	0.001D0	1.000D0
26	ffuspow	f-value for fusion power limit equation	9	0.001D0	1.000D0
27	fhldiv	f-value for divertor heat load limit equation	18	0.001D0	1.000D0
28	fradpwr	f-value for radiation power limit equation	17	0.001D0	1.000D0
29	bore	machine bore		0.100D0	10.00D0
30	fmva	f-value for MVA limit equation	19	0.010D0	1.000D0
31	gapomin	minimum gap between outboard vacuum vessel and TF coil		0.001D0	10.00D0
32	frminor	f-value for minor radius limit equation	21	0.001D0	1.000D0
33	fportsz	f-value for beam tangency radius limit equation	20	0.001D0	1.000D0
34	fdivcol	f-value for divertor collisionality limit equation	22	0.001D0	1.000D0
35	fpeakb	f-value for peak toroidal field limit equation	25	0.001D0	1.000D0
36	fbetatry	f-value for beta limit equation	24	0.001D0	1.000D0
37	coheof	central solenoid current density at end of flat-top		1.000D5	1.000D8
38	fjohc	f-value for central solenoid current at EOF limit equation	26	0.010D0	1.000D0
39	fjohc0	f-value for central solenoid current at BOP limit equation	27	0.001D0	1.000D0
40	fgamcd	f-value for current drive gamma limit equation	37	0.001D0	1.000D0
41	fcohbop	central solenoid current density ratio BOP/EOF		0.001D0	1.000D0
42	gapoh	gap between central solenoid and TF coil		0.001D0	10.00D0
43	cfe0	seeded high-Z impurity fraction		1.00D-6	3.00D-3
44	fvsbrnni	fraction of plasma current produced by non-inductive means		0.001D0	1.000D0
45	fqval	f-value for fusion gain limit equation	28	0.001D0	1.000D0
46	fpinj	f-value for injection power limit equation	30	0.001D0	1.000D0
47	feffcd	current drive efficiency multiplier		0.001D0	1.000D0
48	fstrcase	f-value for TF coil case stress limit equation	31	0.001D0	1.000D0
49	fstrcond	f-value for TF coil conduit stress limit equation	32	0.001D0	1.000D0
50	fiooic	f-value for TF coil operational current limit equation	33	0.001D0	0.500D0

Table 2.2: *Iteration variables 1 to 50 present in* PROCESS*. The f-values correspond to the given constraint equations (see Table 2.1). The other iteration variables are shown in Table 2.3.*

ixc no.	variable name	description	icc eqn	lower bound	upper bound
51	fvdump	f-value for TF coil dump voltage limit equation	34	0.001D0	1.000D0
52	vdalw	allowable TF coil dump voltage		0.001D0	1.000D6
53	fjprot	f-value for TF coil current protection limit equation	35	0.001D0	1.000D0
54	ftmargtf	f-value for TF coil temperature margin limit equation	36	0.001D0	1.000D0
55	tmargmin	minimum allowable TF coil temperature margin		0.001D0	100.0D0
56	tdmptf	dump time for TF coil		10.00D0	1.000D6
57	thkcas	TF coil external case thickness		0.050D0	1.000D0
58	thwcndut	TF coil conduit case thickness		0.001D0	1.000D0
59	fcutfsu	copper fraction of cable conductor		0.001D0	1.000D0
60	cpttf	current per turn in the TF coils		0.001D0	4.000D4
61	gapds	gap between vacuum vessel and inboard TF coil		0.001D0	10.00D0
62	fdtmp	f-value for 1st wall coolant temperature rise limit equation	38	0.001D0	1.000D0
63	ftpeak	f-value for 1st wall peak temperature limit equation	39	0.001D0	1.000D0
64	fauxmn	f-value for minimum auxiliary power limit equation	40	0.001D0	1.000D0
65	tohs	central solenoid current ramp-up time		0.100D0	1.000D3
66	ftohs	f-value for central solenoid current ramp-up time limit equation	41	0.001D0	1.000D0
67	ftcycl	f-value for minimum cycle time limit equation	42	0.001D0	1.000D0
68	fptemp	f-value for maximum centrepost temperature limit equation	44	0.001D0	1.000D0
69	rcool	average radius of centrepost coolant channel		0.001D0	0.010D0
70	vcool	maximum centrepost coolant flow speed at midplane		1.000D0	1.000D2
71	fq	f-value for minimum edge safety factor limit equation	45	0.001D0	1.000D0
72	fipir	f-value for maximum I_p/I_{rod} limit equation	46	0.001D0	1.000D0
73	scrapli	inboard gap between plasma and first wall		0.001D0	10.00D0
74	scraplo	outboard gap between plasma and first wall		0.001D0	10.00D0
75	tfootfi	ratio of TF coil outboard/inboard leg thickness		0.200D0	5.000D0
76	frfptf	f-value for TF coil toroidal thickness limit equation	47	0.001D0	1.000D0
77	tftort	TF coil toroidal thickness (use for RFPs only)		0.050D0	4.000D0
78	rfpth	RFP pinch parameter, Θ		0.010D0	1.800D0
79	fbetap	f-value for poloidal beta limit equation	48	0.001D0	1.000D0
80	frfpf	f-value for RFP reversal parameter limit equation	49	0.001D0	1.000D0
81	edrive	IFE driver energy		1.000D5	5.000D7
82	drveff	IFE driver wall plug to target efficiency		0.010D0	1.000D0
83	tgain	IFE target gain		1.000D0	500.0D0
84	chrad	radius of IFE chamber		0.100D0	20.00D0
85	pdrive	IFE driver power reaching target		1.000D6	2.000D8
86	frrmax	f-value for maximum IFE repetition rate equation	50	0.001D0	1.000D0
87	helecmw	electrical power required for hydrogen production		1.000D0	4.000D3
88	hthermmw	thermal power required for hydrogen production		1.000D0	4.000D3
89	ftbr	f-value for tritium breeding ratio limit equation	52	0.001D0	1.000D0
90	blbuith	inboard blanket breeding unit thickness		0.001D0	2.000D0
91	blbuoth	outboard blanket breeding unit thickness		0.001D0	2.000D0
92	fflutf	f-value for fast neutron fluence on TF coil equation	53	0.001D0	1.000D0
93	shldith	inboard shield thickness		0.001D0	10.00D0
94	shldoth	outboard shield thickness		0.001D0	10.00D0
95	fptfnuc	f-value for TF coil nuclear heating limit equation	54	0.001D0	1.000D0
96	fvvhe	f-value for vessel He concentration limit equation	55	0.001D0	1.000D0
97	fpsepr	f-value for $P_{\text{separatrix}}/R_{\text{major}}$ limit equation	56	0.001D0	1.000D0
98	li6enrich	lithium-6 enrichment percentage (blktmodel=1)		0.001D0	100.0D0
99	ftftort	(OBSOLETE) f-value for TF coil toroidal thickness lower limit eqn	57	0.001D0	1.000D0
100	ftfthko	(OBSOLETE) f-value for TF coil radial thickness lower limit eqn	58	0.001D0	1.000D0
101	prp	ratio of TF coil radial plate area to winding pack area		1.00D-6	0.010D0
102	fimpvar	impurity fraction of element impvar		1.00D-6	0.010D0

Table 2.3: *Iteration variables 51 to 102 present in PROCESS. The f-values correspond to the given constraint equations (see Table 2.1). The other iteration variables are shown in Table 2.2.*

`minmax`	description
±1	plasma major radius
±2	ratio of fusion power to input power
±3	neutron wall load
±4	total TF coil + PF coil power
±5	ratio of fusion power to injection power
±6	cost of electricity
±7	{ direct cost if `ireactor = 0`; constructed cost otherwise
±8	aspect ratio
±9	divertor heat load
±10	toroidal field on axis
±11	injection power
±12	hydrogen plant capital cost
±13	hydrogen production rate
±14	pulse length
±15	plant availability factor (N.B. requires `iavail=1`)

Table 2.4: *Summary of the available figures of merit in PROCESS. If the figure of merit is to be minimised, minmax should be positive, and if a maximised figure of merit is desired, minmax should be negative.*

physics and engineering issues in terms of numerous expressions and relationships. In effect these define the machine so that the numerical solver within the code can then get to work adjusting the parameters in its search for a self-consistent solution.

It is essential for a program of the size and complexity of PROCESS to be modular to a high degree, in order to simplify the tasks of understanding and maintaining the code. The use of Fortran 90/95 modules provides a natural and convenient way for this to be done. The following sections describe briefly the modules into which PROCESS is divided.

2.8.1 Numerics modules

These modules contain the equation solvers, their calling routines and other relevant procedures. Various mathematical routines from a number of standard libraries are also incorporated into these files. Table 2.6 summarises the numerics source file contents.

2.8.2 Physics modules

These modules contain the main physics routines that evaluate the plasma and fusion parameters. Also included here are the routines describing the current drive and divertor systems. Table 2.7 summarises the physics source file contents.

2.8.3 Engineering modules

These modules contain the description of the machine geometry and its major systems, including the PF and TF coil sets, the first wall, blanket and shield, and other items such as the buildings, vacuum system, power conversion and the structural components. Table 2.8 summarises the engineering source

nsweep	scan variable	description
1	aspect	plasma aspect ratio
2	hldivlim	maximum divertor heat load
3	pnetelin	required net electric power
4	hfact	confinement time H-factor
5	oacdcp	overall current density in TF coil inboard leg
6	walalw	allowable wall load
7	beamfus0	beam-background fusion multiplier
8	fqval	f-value for fusion gain limit eqn
9	te	electron temperature
10	boundu(15)	upper bound on f-value fvs
11	dnbeta	beta g coefficient
12	bscfmax	bootstrap current fraction (use negative values)
13	boundu(10)	upper bound on confinement time H-factor hfact
14	fiooic	f-value for TF coil operational current limit eqn
15	fjprot	f-value for TF coil current protection limit eqn
16	rmajor	plasma major radius
17	bmxlim	maximum toroidal field
18	gammax	maximum current drive gamma
19	boundl(16)	lower bound on central solenoid thickness ohcth
20	tbrnmn	minimum burn time (pulsed operation machine)
21	sigpfalw	allowable stress in the PF coils
22	cfactr	plant availability factor (N.B. requires iavail=0)
23	boundu(72)	upper bound on f-value fipir
24	powfmax	maximum fusion power
25	kappa	plasma elongation
26	triang	plasma triangularity
27	tbrmin	minimum tritium breeding ratio (blktmodel=1)
28	bt	toroidal field on axis
29	coreradius	normalised radius defining the core region
30	fimpvar	impurity fraction of element impvar

Table 2.5: *Summary of the scanning variables available in PROCESS.*

source file	description
caller.f90	calls physics and engineering routines
constraint_equations.f90	defines the constraint equations
evaluators.f90	function evaluators for HYBRD and VMCON packages
iteration_variables.f90	adjusts values of iteration variables
maths_library.f90	miscellaneous 'black-box' maths routines, including HYBRD and VMCON
numerics.f90	numerics array definitions, and calling routines for HYBRD and VMCON packages
scan.f90	performs a parameter scan

Table 2.6: *Summary of the numerics modules in PROCESS.*

source file	description
`current_drive.f90`	current drive efficiency calculations
`divertor.f90`	divertor model calculations
`fispact.f90`	nuclide inventory/activation calculations
`ife.f90`	inertial fusion energy physics/engineering
`impurity_radiation.f90`	radiation power calculations
`physics.f90`	tokamak plasma and fusion calculations
`plasma_geometry.f90`	plasma geometry algorithms
`plasma_profiles.f90`	plasma density and temperature profile calculations
`rfp.f90`	reversed field pinch physics/engineering
`startup.f90`	plasma start-up auxiliary power requirements
`stellarator.f90`	stellarator-relevant physics/engineering

Table 2.7: *Summary of the physics modules in* `PROCESS`.

file contents.

source file	description
`availability.f90`	plant component lifetimes and overall availability
`buildings.f90`	buildings calculations
`fwbs.f90`	first wall, blanket and shield calculations
`machine_build.f90`	machine build calculations
`pfcoil.f90`	PF coil calculations
`plant_power.f90`	heat transport and power balance calculations
`pulse.f90`	pulsed power plant calculations
`safety.f90`	steady-state temperatures after a LOCA event
`sctfcoil.f90`	superconducting TF coil calculations
`structure.f90`	support structure calculations
`tfcoil.f90`	resistive TF coil calculations
`vacuum.f90`	vacuum system calculations

Table 2.8: *Summary of the engineering modules in* `PROCESS`.

2.8.4 Costing module

The costing module, `costs.f90`, performs all the cost calculations, including values in M$ for each machine system, and the cost of electricity in m$/kWh.

2.8.5 Other modules

These modules perform miscellaneous tasks, such as initialisation of variables and file input / output. File `process.f90` contains the main program, and includes the overall controlling loop.

Table 2.9 summarises these modules.

source file	description
`error_handling.f90`	centralised error handling module
`fson_library.f90`	library used to read in data from JSON-format files
`global_variables.f90`	defines and initialises most shared variables
`initial.f90`	checks self-consistency of input variables and switches
`input.f90`	reads in user-defined settings from input file
`output.f90`	utility routines to format output to file
`process.f90`	main program and top-level calling routines

Table 2.9: *Summary of the remaining modules in* `PROCESS`.

Chapter 3

Physics, Engineering and Other Models

There are a great number of individual models within PROCESS, characterising many different aspects of a fusion power plant. Several of these will always be used by the code and so require no input by the user to activate them. However, in many cases there is a choice of model available, and each of these has its own user-controlled switches or flags. This chapter summarises these models, and indicates their location and interaction within the code, together with the relevant switch settings and required parameter values. The concepts used (iteration variables, constraint equations, etc.), and instructions on how to set switches, etc., are explained fully in Chapter 2.

Disclaimer: Users are discouraged from using non-default or older models, as they are likely to be less well tested than the default models and should therefore be treated with caution.

3.1 Tokamak Power Plant

The default (and most detailed) power plant model in PROCESS is based on the tokamak magnetic-confinement fusion concept. This section describes the models relevant to such a plant in some detail, starting with the plasma at the centre of the fusion power core, and moving outwards to the external components, power subsystems and buildings. However, many of these models are also partly or wholly relevant for the other available machine types, especially outside the fusion power core. The specific details for these alternative models are introduced later in the Chapter.

3.1.1 Radial and vertical build

Figure 3.1 shows schematically the layout of a typical tokamak as modelled by PROCESS. This is the so-called 'build' of the machine — the relative locations of the major components. Their positions are referenced to the (R, Z) coordinate system, where R is the radial distance from the vertical centreline (axis) of the torus, and Z is the vertical distance from the equatorial midplane, about which the machine is assumed to be up-down symmetrical (by default; the vertical build is slightly different for single null plasma devices — see Figure 3.3). Components are often referred to as being 'inboard' or 'outboard', which simply means that they lie at a radius R less than or greater than R_0, respectively, where R_0 is the plasma major radius (`rmajor`). For the sake of clarity the thicknesses are not drawn to scale, and the space labelled as the divertor does not indicate in any way the actual shape of that component.

The first wall, blanket, shield and vacuum vessel may be either D-shaped in cross-section, or each may

be defined by two half-ellipses; compare their shapes in Figures 3.1 and 3.2. The choice between these two possibilities is set using input parameter `fwbsshape` [8], which should be `1` for D-shaped, or `2` for ellipses.

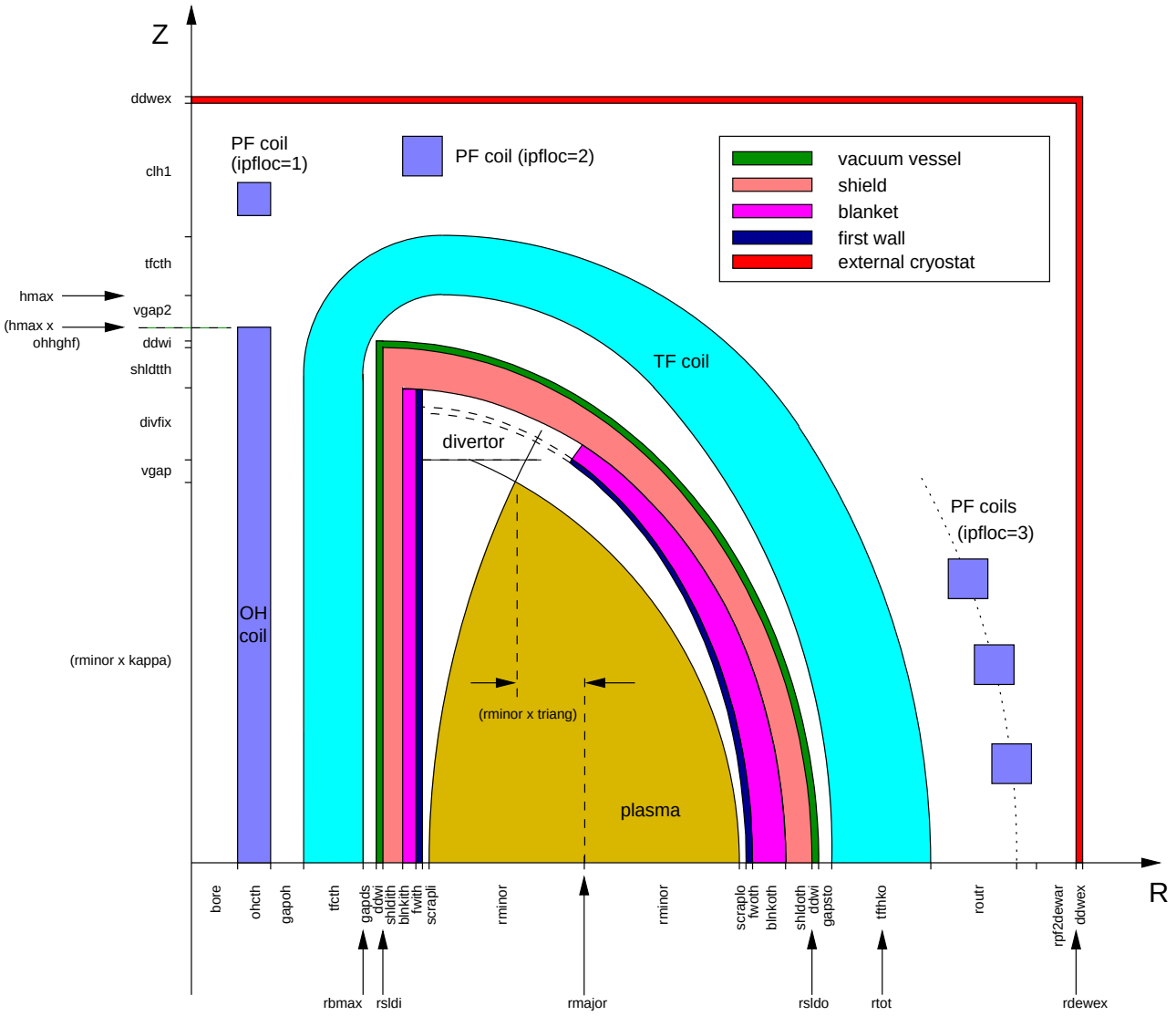

Figure 3.1: *Schematic diagram of the fusion power core of a typical tokamak power plant modelled by* `PROCESS`*, showing the relative positions of the components. A double null plasma is assumed (*`snull=0`*) — compare Figure 3.3, and the first wall, blanket, shield and vacuum vessel are D-shaped in cross-section (chosen by setting switch* `fwbsshape=1`*) — compare Figure 3.2. Also shown are the code variables used to define the thicknesses of the components. The arrowed labels adjacent to the axes are the total 'builds' to that point. The precise locations and sizes of the PF coils are calculated within the code (see Section 3.1.8).*

Most of the thicknesses shown in Figures 3.1 and 3.2 are input parameters, so are not changed during the course of the simulation. The rest are calculated by the code during execution. In addition, some of the component sizes can be used as *iteration variables* (see Section 2.5) to help in the optimisation process.

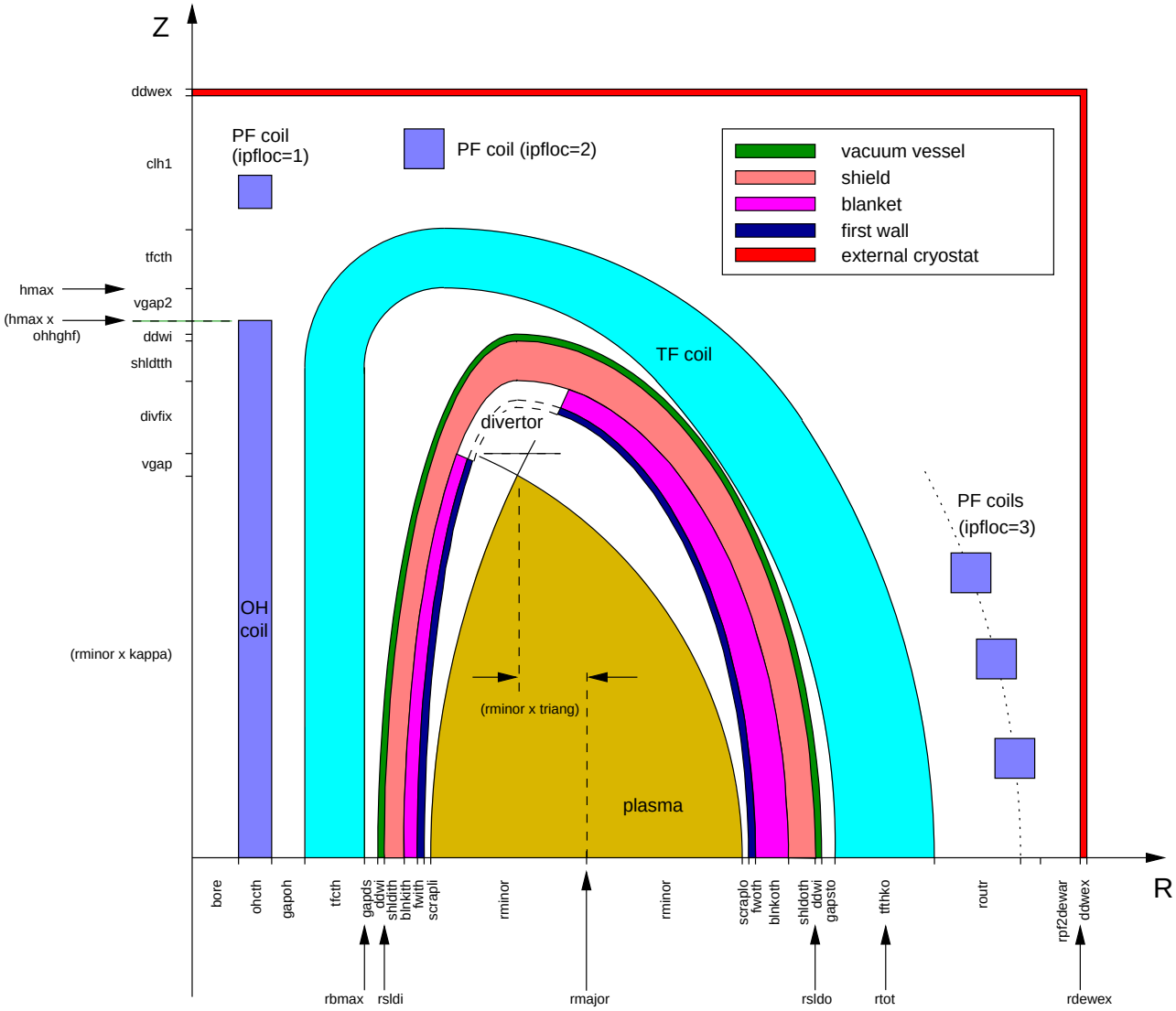

Figure 3.2: *Schematic diagram of the fusion power core of a typical tokamak power plant modelled by* `PROCESS`. *The first wall, blanket, shield and vacuum vessel cross-sectional shapes are each assumed to be defined by two ellipses (chosen by setting switch* `fwbsshape=2`*) — compare Figure 3.1.*

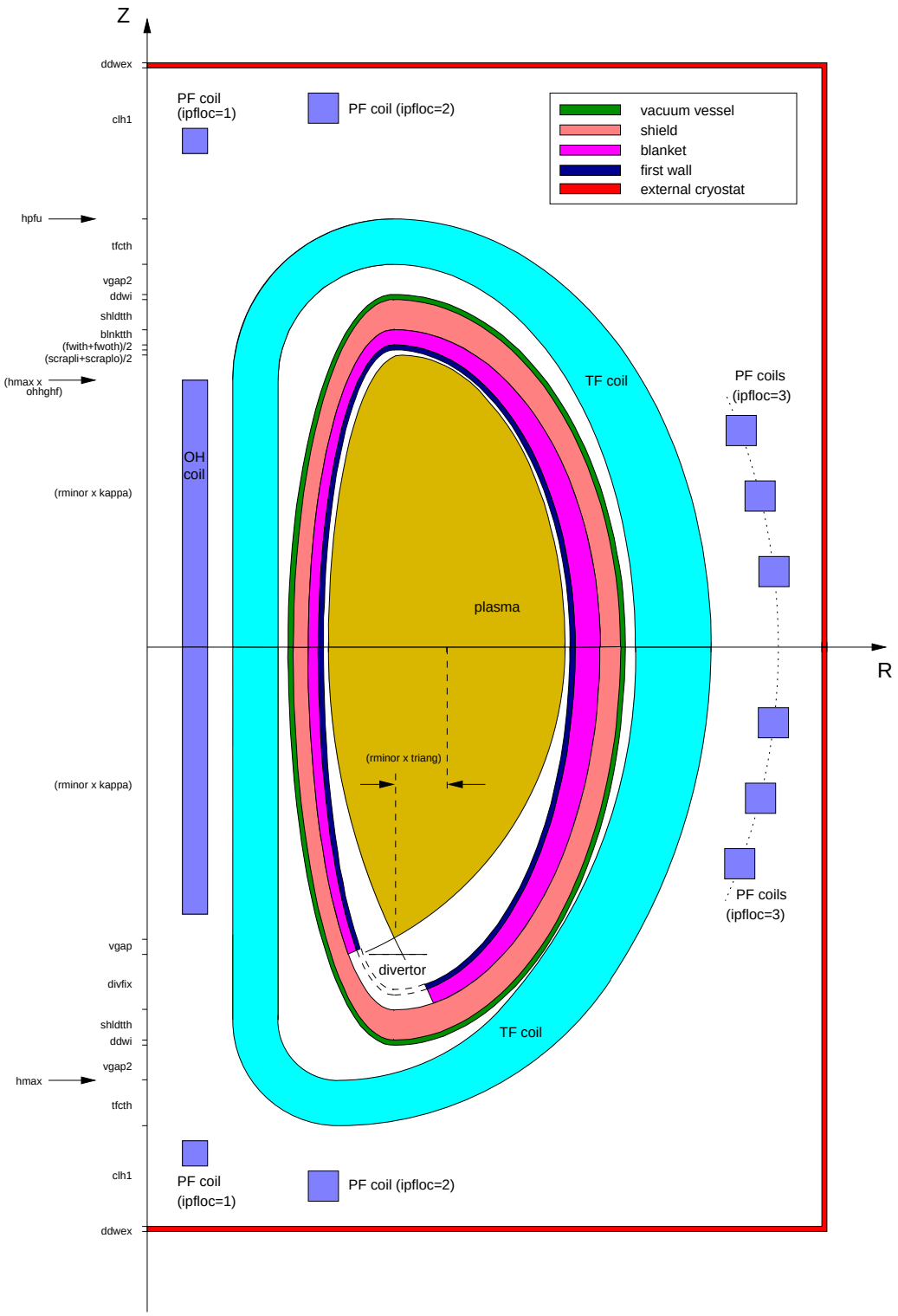

Figure 3.3: *Schematic diagram of the fusion power core of a typical tokamak power plant modelled by* ***PROCESS****, showing the relative positions of the components. A single null plasma is assumed (*`snull=1`*) — compare Figure 3.2. The radial build is the same as for a double null configuration; shown along the vertical axis are the code variables used to define the vertical thicknesses of the components. The arrowed labels adjacent to the axis are the total 'builds' (distance from the midplane, Z=0) to that point. The precise locations and sizes of the PF coils are calculated within the code (see Section 3.1.8).*

3.1.2 Plasma physics models

Arguably, the most important component of the machine is the plasma itself. By default, this is assumed to have an up-down asymmetric, single null configuration (although this can be changed if desired — see Section 3.1.4). A great number of physics models are coded within `PROCESS` to describe the behaviour of the plasma parameters such as its current, temperature, density, pressure, confinement etc., and also the various limits that define the stable operating domain.

3.1.2.1 Plasma geometry

The plasma (geometric) major radius R_0 (`rmajor`) and aspect ratio A (`aspect`) define the size of the plasma torus. The plasma minor radius a (`rminor`) is calculated from these values. The shape of the plasma cross-section is given by its last closed flux surface (LCFS) elongation κ (`kappa`) and triangularity δ (`triang`), which can be scaled automatically with the aspect ratio if required using switch `ishape`:

- If `ishape = 0`, the input values for `kappa` and `triang` are used directly.
- If `ishape = 1`, the following scaling is used, which is suitable for low aspect ratio machines ($\epsilon = 1/A$) [2]:

$$\kappa = 2.05\,(1 + 0.44\,\epsilon^{2.1}) \tag{3.1}$$
$$\delta = 0.53\,(1 + 0.77\,\epsilon^{3}) \tag{3.2}$$

- If `ishape = 2`, the Zohm ITER scaling [9] is used to calculate the elongation:

$$\kappa = \text{minimum}\left(2.0,\ 1.5 + \frac{0.5}{A-1}\right) \tag{3.3}$$

  while the input value of the triangularity is used unchanged.

The values for the plasma shaping parameters at the 95% flux surface are calculated as follows [10]:

$$\kappa_{95} = \kappa/1.12 \tag{3.4}$$
$$\delta_{95} = \delta/1.5 \tag{3.5}$$

The plasma surface area, cross-sectional area and volume are calculated using formulations that approximate the LCFS as a revolution of two arcs which intersect the plasma X-points and the plasma midplane outer and inner radii. (This is a reasonable assumption for double-null diverted plasmas, but will be inaccurate for single-null plasmas, `snull = 1`.) Switch `igeom` determines whether an old method is used (`igeom = 0`) or calculations based on a more recent derivation (`igeom = 1`) [11].

3.1.2.2 Fusion reactions

The most likely fusion reaction to be utilised in a power plant is the deuterium-tritium reaction:

$$\mathrm{D} + \mathrm{T} \Longrightarrow {}^{4}\mathrm{He} + \mathrm{n} + 17.6\,\mathrm{MeV} \tag{3.6}$$

20% of the energy produced is given to the alpha particles (^{4}He), which remain (but c.f. falpha) within the plasma and thermalise (slow down) due to collisions, thus heating the plasma. The remaining 80% is carried away by the neutrons, which deposit their energy within the blanket and shield.

PROCESS can also model D-^{3}He power plants, which utilise the following primary fusion reaction:

$$\mathrm{D} + {}^3\mathrm{He} \Longrightarrow {}^4\mathrm{He} + \mathrm{p} + 18.3\,\mathrm{MeV} \tag{3.7}$$

The fusion reaction rate is significantly different to that for D-T fusion, and the power flow from the plasma is modified since charged particles are produced rather than neutrons. Because only charged particles (which remain in the plasma) are produced by this reaction, the whole of the fusion power is used to heat the plasma. Useful energy is extracted from the plasma since the radiation power produced is very high, and this can be converted to electricity in a number of ways.

Since the temperature required to ignite the D-^{3}He reaction is considerably higher than that for D-T, it is necessary to take into account the following D-D reactions, which have significant reaction rates at such temperatures:

$$\mathrm{D} + \mathrm{D} \Longrightarrow {}^3\mathrm{He} + \mathrm{n} + 3.27\,\mathrm{MeV} \tag{3.8}$$

$$\mathrm{D} + \mathrm{D} \Longrightarrow \mathrm{T} + \mathrm{p} + 4.03\,\mathrm{MeV} \tag{3.9}$$

Also, as tritium is produced by the latter reaction, D-T fusion is also possible. As a result, there is still a small amount of neutron power extracted from the plasma.

Pure D-^{3}He tokamak power plants do not include blankets, because of the near absence of neutrons leaving the plasma, and the fact that no tritium needs to be produced for fuel.

The contributions from all four of the above fusion reactions are included in the total fusion power production calculation. The fusion reaction rates are calculated using the parametrizations in [12], integrated over the plasma profiles (correctly, with or without pedestals).

The fractional composition of the "fuel" ions (D, T and ^{3}He) is controlled using the three variables `fdeut`, `ftrit` and `fhe3`, respectively:

$$\begin{aligned} n_{\mathrm{fuel}} &= n_D + n_T + n_{^3\mathrm{He}} \quad \text{particles/m}^3 \\ n_D &= \texttt{fdeut}\, n_{\mathrm{fuel}} \\ n_T &= \texttt{ftrit}\, n_{\mathrm{fuel}} \\ n_{^3\mathrm{He}} &= \texttt{fhe3}\, n_{\mathrm{fuel}} \end{aligned}$$

PROCESS checks that `fdeut + ftrit + fhe3 = 1.0`, and stops with an error message otherwise.

3.1.2.3 Plasma profiles

By default NOT ANY MORE! (switch `ipedestal = 0`), the plasma profiles are assumed to be parabolic, i.e. they are of the form

$$\text{Density} : n(\rho) = n_0 \left(1 - \rho^2\right)^{\alpha_n} \tag{3.10}$$

$$\text{Temperature} : T(\rho) = T_0 \left(1 - \rho^2\right)^{\alpha_T} \tag{3.11}$$

$$\text{Current} : J(r) = J_0 \left(1 - \rho^2\right)^{\alpha_J} \tag{3.12}$$

where $\rho = r/a$, and a is the plasma minor radius. This gives volume-averaged values $\langle n \rangle = n_0/(1+\alpha_n)$, and line-averaged values $\bar{n} \sim n_0/\sqrt{(1+\alpha_n)}$, etc. These volume- and line-averages are used throughout the code along with the profile indices α, in the various physics models, many of which are fits to

theory-based or empirical scalings. Thus, the plasma model in `PROCESS` may be described as "$\frac{1}{2}$-D". The relevant profile index variables are `alphan`, `alphat` and `alphaj`, respectively (see Section 3.1.2.8).

If `ipedestal` is set to `1`, however, the density and temperature profiles may include a pedestal, using the forms specified in [13]:

$$\text{density:} \qquad n(\rho) = \begin{cases} n_{ped} + (n_0 - n_{ped}) \left(1 - \dfrac{\rho^2}{\rho_{ped,n}^2}\right)^{\alpha_n} & 0 \le \rho \le \rho_{ped,n} \\ n_{sep} + (n_{ped} - n_{sep}) \left(\dfrac{1-\rho}{1-\rho_{ped,n}}\right) & \rho_{ped,n} < \rho \le 1 \end{cases} \tag{3.13}$$

$$\text{temperature:} \qquad T(\rho) = \begin{cases} T_{ped} + (T_0 - T_{ped}) \left(1 - \dfrac{\rho^{\beta_T}}{\rho_{ped,T}^{\beta_T}}\right)^{\alpha_T} & 0 \le \rho \le \rho_{ped,T} \\ T_{sep} + (T_{ped} - T_{sep}) \left(\dfrac{1-\rho}{1-\rho_{ped,T}}\right) & \rho_{ped,T} < \rho \le 1 \end{cases} \tag{3.14}$$

Subscripts 0, *ped* and *sep*, denote values at the centre ($\rho = 0$), the pedestal ($\rho = \rho_{ped}$) and the separatrix ($\rho = 1$), respectively. The density and temperature peaking parameters α_n and α_T as well as the second exponent β_T (input parameter `tbeta`, not to be confused with the plasma beta) in the temperature profile can be chosen by the user, as can the pedestal heights and the values at the separatrix (`neped, nesep` for the electron density, and `teped, tesep` for the electron temperature; the ion equivalents are scaled from the electron values by the ratio of the volume-averaged values). The density at the centre is given by

$$\begin{aligned} n_0 \;=\; & \frac{1}{3\rho_{ped,n}^2} \left[3\langle n \rangle (1+\alpha_n) + n_{sep}(1+\alpha_n)(-2+\rho_{ped,n}+\rho_{ped,n}^2) \right. \\ & \left. - n_{ped} \left((1+\alpha_n)(1+\rho_{ped,n}) + (\alpha_n - 2)\rho_{ped,n}^2 \right) \right] \end{aligned} \tag{3.15}$$

where $\langle n \rangle$ is the volume-averaged density. The temperature at the centre is given by

$$T_0 = T_{ped} + \gamma \left[T_{ped}\, \rho_{ped,T}^2 - \langle T \rangle + \frac{1}{3}(1-\rho_{ped,T}) \left[\left(1 + 2\rho_{ped,T}\right) T_{ped} + \left(2 + \rho_{ped,T}\right) T_{sep} \right] \right] \tag{3.16}$$

with

$$\gamma = \begin{cases} \dfrac{-\Gamma(1+\alpha_T+2/\beta_T)}{\rho_{ped,T}^2\, \Gamma(1+\alpha_T)\, \Gamma((2+\beta_T)/\beta_T)} & \text{for integer } \alpha_T \\ \dfrac{\Gamma(-\alpha_T)\sin(\pi\alpha)\, \Gamma(1+\alpha_T+2/\beta_T)}{\pi \rho_{ped,T}^2\, \Gamma((2+\beta_T)/\beta_T)} & \text{for non-integer } \alpha_T \end{cases} \tag{3.17}$$

where Γ is the gamma function.

Note that both density and temperature can have different pedestal positions $\rho_{ped,n}$ (`rhopedn`) and $\rho_{ped,T}$ (`rhopedt`) in agreement with simulations.

3.1.2.4 Beta limits

The plasma beta limit [14, 15] is given by

$$\langle \beta \rangle < g\, \frac{I(\mathrm{MA})}{a(\mathrm{m})\, B_0(\mathrm{T})} \tag{3.18}$$

where B_0 is the axial vacuum toroidal field, and β is defined with respect to the total equilibrium **B**-field [15]. The beta coefficient g is set using input parameter `dnbeta` (but see Section 3.1.2.8). To

apply the beta limit, constraint equation no. 24 should be turned on with iteration variable no. 36 (`fbetatry`). The limit can be applied to either the total plasma beta, in which case switch `iculbl` should be set to 0, to only the thermal component of the plasma beta, in which case `iculbl` should be set to 1, or to the thermal plus neutral beam components, in which case `iculbl` should be set to 2.

Aspect ratio scaling of beta g coefficient

Switch `gtscale` determines whether the beta g coefficient `dnbeta` should scale with aspect ratio (`gtscale` $\neq$ `0`), or be fixed at the input value (`gtscale = 0`). Note that `gtscale` is over-ridden if `iprofile = 1` (see Section 3.1.2.8).

Limiting $\epsilon.\beta_p$

To apply a limit to the value of $\epsilon.\beta_p$, where $\epsilon = a/R$ is the inverse aspect ratio, constraint equation no. 6 should be turned on with iteration variable no. 8 (`fbeta`). The limiting value of $\epsilon.\beta_p$ should be set using input parameter `epbetmax`.

3.1.2.5 Density limits

Several density limit models [15] are available in `PROCESS`. These are calculated in routine `CULDLM`, which is called by `PHYSICS`. To enforce any of these limits, turn on constraint equation no. 5 with iteration variable no. 9 (`fdene`). In addition, switch `idensl` must be set to the relevant value, as follows:-

`idensl = 1 :` ASDEX model

`idensl = 2 :` Borrass model for ITER, I

`idensl = 3 :` Borrass model for ITER, II

`idensl = 4 :` JET edge radiation model

`idensl = 5 :` JET simplified model

`idensl = 6 :` Hugill-Murakami $M.q$ model

`idensl = 7 :` Greenwald model

3.1.2.6 Impurities and radiation

The impurity radiation model in `PROCESS` can be chosen using the switch `imprad_model`, where `imprad_model = 0` gives the original ITER 1989 model [16], and `imprad_model = 1` uses a multi-impurity model which integrates the radiation contributions over an arbitrary choice of density and temperature profiles [17].

If `imprad_model = 0`, the impurity species and fractions are chosen using input parameters `impc`, `impo`, `fbfe`, `cfe0` and `zfear`; see the variable descriptor file for more details.

If `imprad_model = 1`, the impurity number density fractions relative to the electron density are set using input array `fimp(1 to nimp)`, where `nimp = 14` is the number of impurity species in the radiation model. The available impurities are as follows:

1. Hydrogen (fraction calculated by code)
2. Helium (fraction calculated by code)
3. Beryllium
4. Carbon
5. Nitrogen
6. Oxygen
7. Neon
8. Silicon
9. Argon
10. Iron
11. Nickel
12. Krypton
13. Xenon
14. Tungsten

As stated above, the number density fractions for hydrogen (all isotopes) and helium need not be set, as they are calculated by the code to ensure that plasma quasi-neutrality is maintained, and taking into account the fuel ratios `fdeut`, `ftrit` and `fhe3` (see Section 3.1.2.2), and the alpha particle fraction `ralpne` which may be input by the user.

The impurity fraction of one of the elements listed in array `fimp` may be used as an iteration variable (although it will only have any effect if `imprad_model = 1`). The element to use is specified using input parameter `impvar`, which may be set to a value between 3 and `nimp`, and the initial estimate to use for the element's impurity fraction must be set using iteration variable no. 102 (`fimpvar`).

The synchrotron radiation power [18, 19] is assumed to originate from the plasma core. The wall reflection factor `ssync` may be set by the user.

By changing the input parameter `coreradius`, the user may set the normalised radius defining the core region (`imprad_model = 1` only). Only the impurity and synchrotron radiation from the core region affects the confinement scaling (but see `iradloss` below), while the power flowing into the divertor is assumed to exclude all radiation power.

Add radiation power flow diagram as on p.26, logbook24

New overall power flow diagram should elucidate this...

Constraint equation no. 17 with iteration variable no. 28 (`fradpwr`) ensures that the calculated total radiation power does not exceed the total power available that can be converted to radiation (i.e. the sum of the fusion alpha power, other charged particle fusion power, auxiliary injected power and the ohmic power). This constraint should always be turned on.

Switch `iradloss` may be used to exclude (`iradloss = 1`, default) or include (`iradloss = 0`) the core radiation power in the transport power used in the confinement time and power balance calculations (see Section 3.1.2.10).

3.1.2.7 Plasma current scaling laws

A number of plasma current scaling laws exploiting the inverse relationship between plasma current and edge safety factor q_ψ [15] are available in PROCESS. These are calculated in routine CULCUR, which is called by PHYSICS. Flag icurr must be set to the relevant value, as follows:-

icurr = 1 : Peng analytic fit

icurr = 2 : Peng double null divertor scaling (ST) [2]

icurr = 3 : Simple ITER scaling

icurr = 4 : Revised ITER scaling [20]

icurr = 5 : Todd empirical scaling, I

icurr = 6 : Todd empirical scaling, II

icurr = 7 : Connor-Hastie model

3.1.2.8 Plasma current profile consistency

Self-consistency between the plasma current profile parameters [10] can be enforced by setting switch iprofile to 1. This ensures that the current profile peaking factor alphaj is consistent with the input values for the safety factor on axis and at the plasma edge (q0 and q, respectively), the plasma internal inductance l_i is consistent with this alphaj, and the beta g coefficient dnbeta scales with l_i.

It is recommended that current scaling law icurr = 4 is used if iprofile = 1. Switch gtscale is over-ridden if iprofile = 1.

3.1.2.9 Confinement time scaling laws

The particle transport loss power in Watts/m^3 from the plasma is defined as

$$P_{\text{loss}} = \frac{W}{\tau_E} \tag{3.19}$$

where the plasma stored energy per unit volume W is given by

$$W = \frac{3}{2} n\, e\, T$$

with particle number density n in m^{-3} and particle temperature T in eV. PROCESS assumes that the electron and ion energy confinement times τ_E are equal.

Many energy confinement time scaling laws are present within PROCESS, for tokamaks, RFPs or stellarators. These are calculated in routine PCOND. The value of isc determines which of the scalings is used in the plasma energy balance calculation (Section 3.1.2.10). Table 3.1 summarises the available scaling laws.

isc	scaling law	reference
1	Neo-Alcator (ohmic)	[14]
2	Mirnov (H-mode)	[14]
3	Merezhkin-Muhkovatov (L-mode)	[14]
4	Shimomura (H-mode)	JAERI-M 87-080 (1987)
5	Kaye-Goldston (L-mode)	Nuclear Fusion **25** (1985) p.65
6	ITER 89-P (L-mode)	Nuclear Fusion **30** (1990) p.1999
7	ITER 89-O (L-mode)	[14]
8	Rebut-Lallia (L-mode)	Plasma Physics and Controlled Nuclear Fusion Research **2** (1987) p. 187
9	Goldston (L-mode)	Plas. Phys. Controlled Fusion **26** (1984) p.87
10	T10 (L-mode)	[14]
11	JAERI-88 (L-mode)	JAERI-M 88-068 (1988)
12	Kaye-Big Complex (L-mode)	Phys. Fluids B **2** (1990) p.2926
13	ITER H90-P (H-mode)	
14	ITER Mix (minimum of 6 and 7)	
15	Riedel (L-mode)	
16	Christiansen et al. (L-mode)	JET Report JET-P (1991) 03
17	Lackner-Gottardi (L-mode)	Nuclear Fusion **30** (1990) p.767
18	Neo-Kaye (L-mode)	[14]
19	Riedel (H-mode)	
20	ITER H90-P (amended)	Nuclear Fusion **32** (1992) p.318
21	Large Helical Device (stellarator)	Nuclear Fusion **30** (1990) p.11
22	Gyro-reduced Bohm (stellarator)	Bull. Am. Phys. Society, **34** (1989) p.1964
23	Lackner-Gottardi (stellarator)	Nuclear Fusion **30** (1990) p.767
24	ITER-93H (H-mode)	Plasma Physics and Controlled Nuclear Fusion Research (Proc. 15th Int. Conf., Seville, 1994) IAEA-CN-60/E-P-3
25	TITAN (RFP)	TITAN RFP Fusion Reactor Study, Scoping Phase Report UCLA-PPG-1100, page 5–9, Jan 1987
26	ITER H-97P ELM-free (H-mode)	J. G. Cordey et al., EPS Berchtesgaden, 1997
27	ITER H-97P ELMy (H-mode)	J. G. Cordey et al., EPS Berchtesgaden, 1997
28	ITER-96P (= ITER97-L) (L-mode)	Nuclear Fusion **37** (1997) p.1303
29	Valovic modified ELMy (H-mode)	
30	Kaye PPPL April 98 (L-mode)	
31	ITERH-PB98P(y) (H-mode)	
32	IPB98(y) (H-mode)	Nuclear Fusion **39** (1999) p.2175
33	IPB98(y,1) (H-mode)	Nuclear Fusion **39** (1999) p.2175
34	IPB98(y,2) (H-mode)	Nuclear Fusion **39** (1999) p.2175
35	IPB98(y,3) (H-mode)	Nuclear Fusion **39** (1999) p.2175
36	IPB98(y,4) (H-mode)	Nuclear Fusion **39** (1999) p.2175
37	ISS95 (stellarator)	Nuclear Fusion **36** (1996) p.1063
38	ISS04 (stellarator)	Nuclear Fusion **45** (2005) p.1684
39	DS03 (H-mode)	Plasma Phys. Control. Fusion **50** (2008) 043001, equation 4.13

Table 3.1: *Summary of the energy confinement time scaling laws in PROCESS.*

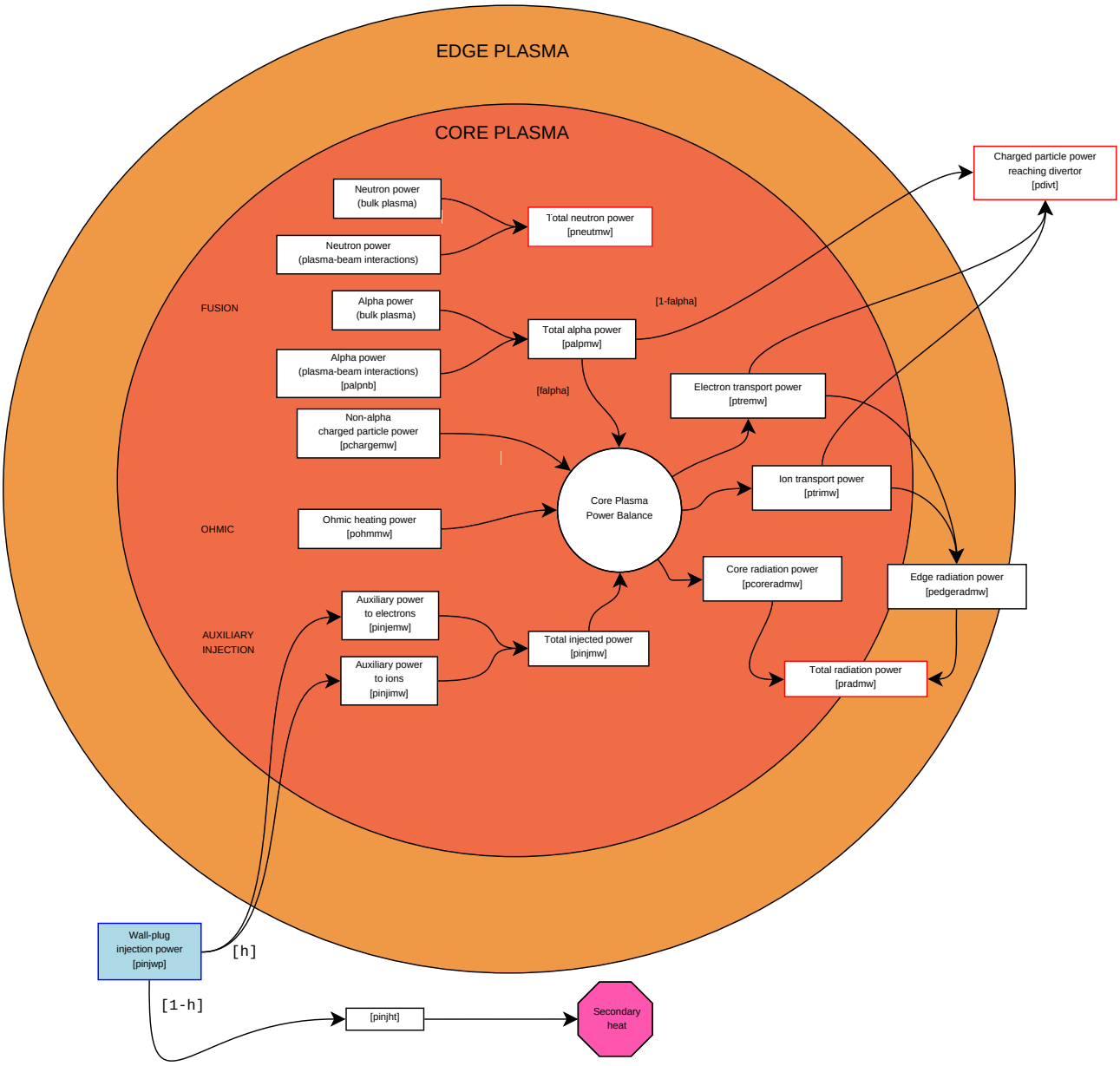

Figure 3.4: *Schematic diagram of the flow of power within the plasma core. Variable names are given in [. . .]. Refer to Figure 3.8 for the onward flow of power from the three red-bordered boxes, while the source of the power in the blue box is indicated in Figure 3.9.*

3.1.2.10 Plasma core power balance

Figure 3.4 shows schematically the flow of power within the plasma as calculated by the code. (Figures 3.8 and 3.9 show the power transfer mechanisms for the remainder of the power plant.)

The primary sources of power are the fusion reactions themselves, ohmic power due to resistive heating within the plasma, and any auxiliary power provided for heating and current drive (see Section 3.1.10). The power carried by the fusion-generated neutrons is lost from the plasma, but is deposited in the surrounding material (see Section 3.1.13). A fraction `falpha` of the alpha particle power is assumed to stay within the plasma core to contribute to the plasma power balance. The sum of this core alpha power, any power carried by non-alpha charged particles, the ohmic power and any injected power, is converted into charged particle transport power (P_{loss} in Equation 3.19) plus core radiation power (see Section 3.1.2.6), as shown in Figure 3.4. (Note that, if switch `iradloss` is set to zero, the above summed input power is converted into charged particle transport power only, and the core radiation power may be assumed to originate from the transport power; i.e. the core radiation power is treated the same as the edge radiation power in Figure 3.4 instead of having an arrow leading directly to it from the core plasma power balance.)

The core power balance calculation is turned on using constraint equation no. 2 (which should therefore always be used).

3.1.2.11 Bootstrap current scalings

The fraction of the plasma current provided by the so-called bootstrap effect can be either input into the code directly, or calculated using one of four methods, as summarised here. Note that methods `ibss = 1` to `3` use fits to parabolic density and temperature profiles, and do not take into account the existence of pedestals (`ipedestal = 1`), whereas the Sauter et al. scaling (`ibss = 4`) allows general profiles to be used.

- Direct input: To input the bootstrap current fraction directly, set `bscfmax` to (-1) times the required value.
- ITER scaling [14]: To use the ITER scaling method for the bootstrap current fraction, set `ibss = 1`, and `bscfmax` to the maximum required bootstrap current fraction (≤ 1). This method is valid at high aspect ratio only.
- General scaling [21]: To use a more general scaling method, set `ibss = 2`, and `bscfmax` to the maximum required bootstrap current fraction (≤ 1).
- Numerically fitted scaling [22]: To use a numerically fitted scaling method, valid for all aspect ratios, set `ibss = 3`, and `bscfmax` to the maximum required bootstrap current fraction (≤ 1).
- Sauter, Angioni and Lin-Liu scaling [23, 24]: Set `ibss = 4`, and `bscfmax` to the maximum required bootstrap current fraction (≤ 1).

3.1.2.12 L-H power threshold scalings

Transitions from a standard confinement mode (L-mode) to an improved confinement regime (H-mode), called L-H transitions, are observed in most tokamaks. A range of scaling laws are available that provide estimates of the auxiliary power required to initiate these transitions, via extrapolations from present-day devices. `PROCESS` calculates these power threshold values for the scaling laws listed in Table 3.2, in routine `PTHRESH`.

	name	reference
(1)	1996 ITER scaling: nominal	ITER Physics Design Description Document, D. Boucher, p.2-2
(2)	1996 ITER scaling: upper bound	
(3)	1996 ITER scaling: lower bound	
(4)	1997 ITER scaling, excluding elongation	J. A. Snipes, ITER H-mode Threshold Database Working Group, Controlled Fusion and Plasma Physics, 24th EPS Conference, Berchtesgaden, June 1997, vol.21A, part III, p.961
(5)	1997 ITER scaling, including elongation	
(6)	2008 Martin scaling: nominal	Martin et al, 11th IAEA Tech. Meeting on H-mode Physics and Transport Barriers, Journal of Physics: Conference Series **123** (2008) 012033
(7)	2008 Martin scaling: 95% upper bound	
(8)	2008 Martin scaling: 95% lower bound	

Table 3.2: *Summary of the L-H power threshold scalings implemented in PROCESS.*

3.1.2.13 Other plasma physics options

Neo-classical correction effects

Switch `ires` controls whether neo-classical correction effects [25] are included in the calculation of the plasma resistance and ohmic heating power in routine `POHM`, which is called by routine `PHYSICS`. If `ires = 1`, these effects are included. Note that the scaling used is only valid for aspect ratios between 2.5 and 4, and it is possible for the plasma resistance to be wrongly calculated as negative if `ires = 1` and the aspect ratio is too high.

Inverse quadrature in τ_E scaling laws

Switch `iinvqd` determines whether the energy confinement time scaling laws (see Section 3.1.2.9) due to Kaye-Goldston (`isc = 5`) and Goldston (`isc = 9`) should include an inverse quadrature scaling with the Neo-Alcator result (`isc = 1`). A value `iinvqd = 1` includes this scaling.

Plasma / first wall gap

The region directly outside the last closed flux surface of the core plasma is known as the scrape-off layer, and contains no structural material. Plasma entering this region is not confined and is removed by the divertor. `PROCESS` treats the scrape-off layer merely as a gap. Switch `iscrp` determines whether the inboard and outboard gaps should be calculated as 10% of the plasma minor radius (`iscrp = 0`), or set equal to the input values `scrapli` and `scraplo` (`iscrp = 1`).

3.1.3 First wall

The first wall acts as a physical barrier protecting the rest of the machine from the hot plasma. Due to its hostile environment the first wall has only a short lifetime and therefore needs to be replaced regularly. Its stainless steel structure is cooled either by gaseous helium or by pressurised water, as chosen using switch `costr` – see Section 3.1.5.2.

The first wall coolant fraction is given by the value of `fwclfr`, which is calculated if a pulsed power plant is being modelled (`lpulse = 1` — see Section 3.3), and assumed to be the input value otherwise.

Wall load calculation

Switch `iwalld` determines whether the neutron wall load (power per unit area) should be calculated using the plasma surface area (`iwalld = 1`) or the first wall area (`iwalld = 2`) as the denominator. In the former case, input parameter `ffwal` (default value 0.92) can be used to scale the neutron power reaching the first wall.

3.1.4 Divertor

The divertor provides a means of removing plasma reaching the scrape-off layer and heavy ions that are ejected from the first wall. By default, two divertors are assumed in the `PROCESS` tokamak, placed symmetrically above and below the plasma. The principal outputs from the code are the divertor heat load, used to determine its lifetime, and its peak temperature. The divertor is cooled either by gaseous helium or by pressurised water.

Switch `snull` controls the overall plasma configuration. Setting `snull = 0` corresponds to an up-down symmetric, double null configuration (the default), while `snull = 1` assumes a single null plasma with the divertor at the bottom of the machine. The vertical build (see Figure 3.3) and PF coil current scaling algorithms take the value of this switch into account, although not the plasma geometry at present.

The Harrison-Kukushkin-Hotston divertor model [14] developed for ITER is used (except for the case of spherical tokamaks or stellarators — see later). This is appropriate for conventional aspect ratio machines, but care should be taken in inputting the divertor magnetics for this model, and projections far from the ITER CDA machine parameters are likely to be unreliable.

3.1.5 Blanket

The blanket performs a number of tasks. An incoming neutron from a deuterium-tritium (D-T) fusion reaction in the plasma loses energy in the blanket. This energy is removed by the blanket coolant and used to produce electricity. The neutron may also react with a lithium nucleus present in the blanket to produce ("breed") a tritium nucleus which can be re-used as fuel. The competing requirements of heating and tritium synthesis mean that a neutron multiplier must be present, to ensure balance between tritium destruction and creation. The blanket therefore contains beryllium to fulfil this purpose. As with the first wall, the blanket has a relatively short lifetime because of the high neutron fluence.

3.1.5.1 Blanket neutronics model

Switch `blktmodel` determines whether a simple blanket neutronics model or a more comprehensive treatment based on recent full neutronics analyses for EFDA [26] is used.

In the simple model (`blktmodel = 0`), the overall radial thicknesses of the inboard and outboard blanket sections are specified via input parameters `blnkith` and `blnkoth`, respectively. The energy multiplication in the blanket (`emult`) is also an input parameter. Steel and vanadium may be used as structural materials within the blanket, which is cooled either by gaseous helium or by pressurised water. The nuclear heating of the TF coil superconductor is calculated assuming an exponential neutron attenuation through the blanket and shield materials, based on 1990 ITER data.

The more advanced model (`blktmodel = 1`) allows the energy multiplication factor `emult`, the shielding requirements and tritium breeding ratio to be calculated self-consistently with the blanket

and shielding materials and sub-assembly thicknesses, and for constraints to be applied to satisfy the engineering requirements. The rest of this section describes this model in more detail.

The model is based on the Helium-Cooled Pebble Bed (HCPB) blanket concept developed by KIT (a second advanced model — Helium-Cooled Lithium Lead, HCLL — will be implemented in due course). The blanket, shield and vacuum vessel are segmented radially into a number of sub-assemblies. Moving in the direction away from the plasma/first wall, these are:

- Breeding Zone (BZ) (which includes the first wall), with radial thicknesses (inboard and outboard, respectively) `fwith+blbuith`, `fwoth+blbuoth`. This consists of beryllium (with fraction by volume `fblbe`), breeder material (`fblbreed`), steel (`fblss`) and helium coolant. Three forms of breeder material are available: lithium orthosilicate (Li_4SiO_4) (chosen by setting `breedmat = 1`), lithium metatitanate (Li_2TiO_3) (`breedmat = 2`) or lithium zirconate (Li_2ZrO_3) (`breedmat = 3`). The ^{6}Li enrichment percentage may be modified from the default 30% using input parameter `li6enrich`.

- Box Manifold (BM), with radial thicknesses (inboard and outboard, respectively) `blbmith`, `blbmoth` and helium fractions `fblhebmi`, `fblhebmo` (the rest being steel).

- Back Plate (BP), with radial thicknesses (inboard and outboard, respectively) `blbpith`, `blbpoth` and helium fractions `fblhebpi`, `fblhebpo` (the rest being steel).

  Together, the BZ, BM and BP make up the 'blanket', with total radial thicknesses `blnkith` (inboard) and `blnkoth` (outboard), and void (coolant) fraction `vfblkt`; Note that these quantities are *calculated* from the sub-assembly values if `blktmodel > 0`, rather than being input parameters.

- Low Temperature Shield and Vacuum Vessel (lumped together for these calculations), with radial thicknesses (inboard and outboard, respectively) `shldith+ddwi`, `shldoth+ddwi` and **water** coolant fraction `vfshld` (the rest being assumed to be steel for its mass calculation; the neutronics model assumes that the shield contains 2% boron as a neutron absorber, but this material is not explicitly mentioned elsewhere in the code — so its cost is not calculated, for example).

  N.B. The fact that water is assumed to be the coolant in the shield, whereas helium is the coolant in the blanket, leads to an inconsistency when specifying the coolant type via switch `costr` (see Section 3.1.5.2). At present we mitigate this by forcing `costr=2` (making water the coolant), as in this case the coolant mass and pumping costs are higher, giving the more pessimistic solution with regards to costs.

A few other input parameters are useful for tuning purposes, as follows:

- `fhole` is the 'hole' area fraction of the first wall and blanket that neutrons leaving the plasma see, due to the presence of the divertor, for instance. That is, `1-fhole` is the first wall area coverage factor. (N.B. see Section 3.1.13.)

- `fvolsi` and `fvolso` are the area (and volume) coverage factors for the inboard and outboard shields, respectively.

- `fvoldw` is a multiplier for the volume of the vacuum vessel, used in the item's mass calculation to account for ports, etc.

- `npdiv` is the number of divertor ports, used in the calculation of the tritium breeding ratio (`blktmodel > 0` only).

- `nphcdin` and `nphcdout` are the number of heating/current drive ports on the inboard and outboard sides, respectively, used in the calculation of the tritium breeding ratio (`blktmodel > 0` only). These may be either 'small' (`hcdportsize = 1`) or 'large' (`hcdportsize = 2`).
- `wallpf` is the neutron wall load peaking factor (maximum/mean), used in the calculation of the blanket lifetime (`blktmodel > 0` only).
- `ucblbreed` is the unit cost ($/kg) of the breeder material (`blktmodel > 0` only).

Model outputs and available constraints

The advanced blanket neutronics model provides the following outputs:

- The total nuclear power deposited in the blanket and shield, `pnucblkt` and `pnucshld`, respectively, and the energy multiplication factor in the blanket, `emult` are calculated.
- The tritium breeding ratio, `tbr`. This can be constrained to be no less than a certain value `tbrmin` for tritium self-sufficiency by turning on constraint equation no. 52 with iteration variable no. 89 (`ftbr`). The inboard and outboard blanket BZ thicknesses, `blbuith` and `blbuoth` can also be used as iteration variables (90 and 91, respectively) to help the constraint to be met.
- The tritium production rate in grammes/day is calculated.
- The fast neutron fluence (neutrons/m^2) on the TF coils is calculated. The peak value of this quantity may be constrained to be no more than a maximum value `nflutfmax` by turning on constraint equation no. 53 with iteration variable no. 92 (`fflutf`). The inboard and outboard shield thicknesses, `shldith` and `shldoth` can also be used as iteration variables (93 and 94, respectively) to help the constraint to be met. (Note that in this calculation the TF coil case surrounding the superconductor winding pack is ignored.)
- The nuclear heating power (MW/m^3) on the inboard and outboard TF coils is calculated. Again, this can be limited to be no more than a maximum value `ptfnucmax` by turning on constraint equation no. 54 with iteration variable no. 95 (`fptfnuc`). The inboard and outboard shield thicknesses also help this constraint to be met. (Note that in this calculation the TF coil case surrounding the superconductor winding pack is ignored.) This constraint equation may also be used with `blktmodel = 0`.
- The helium concentration in the vacuum vessel at the end of the plant lifetime is calculated. This needs to be constrained for re-weldability purposes, and can be kept below a maximum value `vvhealw` by turning on constraint equation no. 55 with iteration variable no. 96 (`fvvhe`).
- The blanket lifetime is calculated, assuming a maximum allowable level of neutron damage to its steel of 60 dpa (currently not adjustable). (For the `blktmodel = 0` model, the allowable blanket fluence `abktflnc` in MW-years/m^2 may be input.)

3.1.5.2 Full thermodynamic blanket model

Remove asap.

N.B. it is recommended that this model is *not* used in conjunction with the advanced blanket neutronics model (`blktmodel > 0`).

Switch `lblnkt` determines whether the blanket is to be simulated using a full thermodynamic model [27] (`lblnkt = 1`), or simply assumed to be made up of relevant materials (see Section 3.1.5.3) in user-defined proportions. The former model also performs a self-consistent calculation of the thermal-to-electric conversion efficiency, whereas the latter simply uses the input value `etath`.

The following switches control the details of the full thermodynamic model of the blanket.

- **Cooling channel geometry:** The value of switch `astr` determines whether the cooling channels have a circular cross-section (`astr = 1`) or an annular cross-section (`astr = 2`). The latter case is the default.
- **Cooling channel orientation:** The value of switch `estr` determines whether the cooling channels lie in the radial direction (`estr = 1`) or in the poloidal direction (`estr = 2`). The former case is the default.
- **Coolant type:** The value of switch `costr` determines the type of coolant used in the first wall, blanket and shield. If `costr = 1`, the coolant is assumed to be gaseous helium. If `costr = 2`, the coolant is assumed to be pressurised water/steam, which is the default. This switch is used whether or not the full blanket model is used, i.e. is independent of the setting of switch `lblnkt`.
- **Boundary condition:** The value of switch `bstr` determines whether the coolant output temperature is to be fixed (`bstr = 1`) or whether the maximum blanket temperature is to be fixed (`bstr = 2`). The former case is the default. The desired coolant output temperature for `bstr = 1` is set using input parameter `xtfo`, and the required maximum blanket temperature is set using input parameter `xtb`.

3.1.5.3 Blanket materials

Table 3.3 summarises the possible options for the blanket materials. Switch `smstr` determines whether a solid blanket of Li_2O-Be (`smstr = 1`), or a liquid blanket of LiPb-Li (`smstr = 2`) is used. The former is the default, and is the type assumed if `lblnkt` $\neq$ `1`.

material	`lblnkt` $\neq$ `1`: simple model	`lblnkt = 1`: full thermodynamic model	
		`smstr=1`: solid blanket	`smstr=2`: liquid blanket
stainless steel	`fblss`	`fblss`	`fblss`
vanadium	`fblvd`	`fblvd`	`fblvd`
Li_2O	`fblli2o`	`fblli2o`	—
beryllium	`fblbe`	`fblbe`	—
LiPb	—	—	`fbllipb`
lithium	—	—	`fblli`
coolant	`vfblkt`	`vfblkt`	`vfblkt`

Table 3.3: *Summary of the materials comprising the blanket in the various scenarios available (`blktmodel=0` only). The fractions given are all available to be modified, and should of course add up to 1.0 for any given model. The type of coolant used is given by the value of switch `costr`.*

3.1.6 Shield

The stainless steel shield reduces the neutron flux reaching the TF coils and beyond. This minimises the radiological impact of the neutrons, and their heating of the TF coils which, if superconducting,

need to remain at liquid helium temperatures. The shield is cooled either by gaseous helium or by pressurised water (**but see Section 3.1.5.1**), as chosen using switch `costr` – see Section 3.1.5.2, and as with the blanket the energy deposited in the coolant is used to produce electricity. The shield coolant fraction is stored in input parameter `vfshld`.

The inboard and outboard shield thicknesses (`shldith` and `shldoth`, respectively) may be used as iteration variables.

3.1.7 Toroidal field coils

The toroidal field (TF) coils can be either resistive or superconducting. Switch `itfsup` should be set to `1` for superconducting coils, or `0` for purely copper coils. In the superconductor model, the *CICC* (Conductor In Cable Conduit) structure shown in Figure 3.5 is assumed, and the coils are cooled using a liquid helium cryogenic system.

The steel radial plates shown in the Figure help in the winding process and also provide extra support against tangential stresses. Grooves are machined into the radial plates which act as a guide for the cable. Each winding pack turn slots into a radial plate and is covered by a steel cap. Each turn is assigned half the thickness (along each edge) of its surrounding radial plates and steel caps in the calculation of the cross-sectional areas. Iteration variable no. 101 (`prp`) is the ratio of the total radial plate plus steel cap cross-sectional area within the winding pack to the total winding pack cross-sectional area; from this, the half-thickness `trp` is calculated.

The outboard leg of the TF coil is assumed to be the same width in the toroidal direction as the outside edge of the inboard leg. In the radial direction, for resistive TF coils the input parameter `tfootfi` gives the ratio of the outboard leg thickness to the inboard leg thickness `tfcth`; for superconducting coils the outboard thickness is set equal to the inboard thickness.

Each TF coil is defined in the (R, Z) plane by four circular arcs of different radius, which create a D-shaped profile. Because of the finite number of TF coils used in a tokamak (typically around 16), the toroidal field has a ripple introduced into it, the amplitude of which can be limited to a few percent (given by input parameter `ripmax`, default value 1%) by the code by adjusting the outboard gap thickness (labelled `gapsto` in Figures 3.1 and 3.2).

Among the TF coil parameters calculated by the code are the maximum allowable current density, the stresses on the structure, the energy stored and the magnetic field produced by the coils.

The following options are available within the superconducting TF coil model (`itfsup = 1`).

3.1.7.1 Superconducting materials

Switch `isumattf` specifies which superconducting material is to be used:

`isumattf = 1 :` Nb_3Sn superconductor, ITER critical surface parameterization [28], standard critical values

`isumattf = 2 :` Bi-2212 high temperature superconductor

`isumattf = 3 :` NbTi superconductor

`isumattf = 4 :` Nb_3Sn superconductor, ITER critical surface parameterization [28], user-defined critical parameters

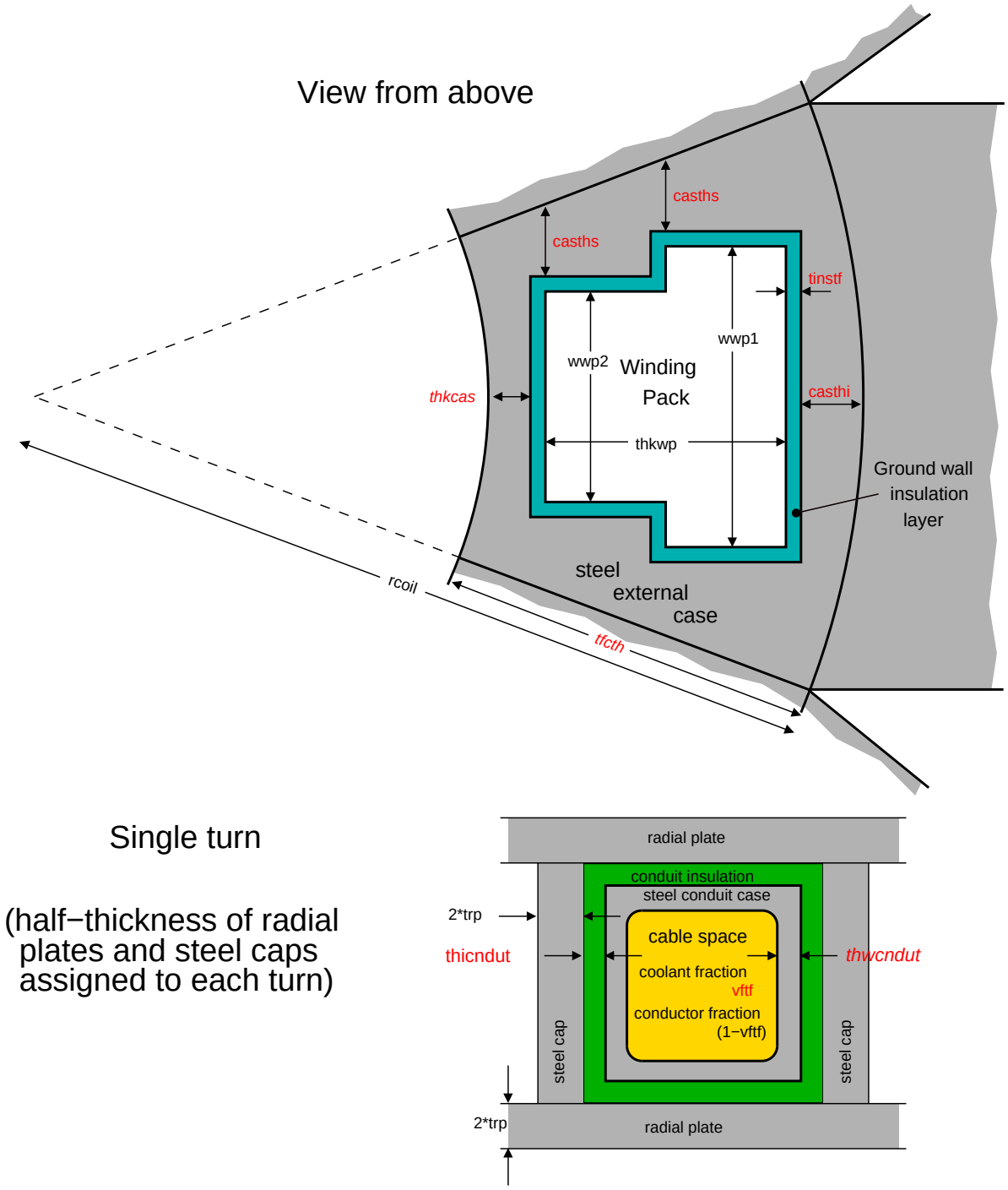

Figure 3.5: *Schematic diagram of the cross-section of the inboard leg of a superconducting TF coil, showing the CICC (Conductor In Cable Conduit) construction. The winding pack contains many turns of cable conduit. The cable space contains the superconducting filaments, and circulating liquid helium coolant. The variables shown in red may be changed by the user, and those in italics may be chosen as iteration variables.*

The fraction of copper present in the superconducting filaments is given by the value of variable `fcutfsu` (iteration variable no. 59).

For `isumattf = 2`, a technology adjustment factor `fhts` may be used to modify the critical current density fit for the Bi-2212 superconductor, to describe the level of technology assumed (i.e. to account for stress, fatigue, radiation, AC losses, joints or manufacturing variations. The default value for `fhts` is 0.5, while a value of 1.0 would be very optimistic.

For `isumattf = 4`, important superconductor properties may be input by the user as follows: the upper critical field at zero temperature and strain is set using input parameter `bcritsc`, and the critical temperature at zero field and strain is set using input parameter `tcritsc`.

3.1.7.2 Stress model

Switch `tfc_model` controls whether a simple stress model (`tfc_model = 0`, suitable for solid copper TF coils) or a more complex stress model (`tfc_model = 1`) should be used. If `tfc_model = 1`, a two-layer stress model [30] developed by CCFE is used.

To enforce the stress limits calculated using either of these models, constraint equation no. 31 (case stress) and/or constraint equation no. 32 (conduit stress) should be turned on with iteration variables no. 48 (`fstrcase`) and/or no. 49 (`fstrcond`), respectively. The stress limit can be adjusted using input parameters `csutf` and `csytf`.

3.1.7.3 Current density limits

The current in the TF coils must be sufficient to produce the required toroidal field at the centre of the plasma. The field falls off at a rate $1/R$, with the peak value occurring at the outer edge of the inboard portion of the TF coil winding pack ($R_{\text{max TF}} =$ `rbmax`). The maximum TF coil current depends on the field it produces and the allowable current density.

Two limits can be applied to the current density J in the (superconducting) TF coils. To ensure that J does not exceed the critical value, constraint equation no. 33 should be turned on with iteration variable no. 50 (`fiooic`). To ensure that J does not exceed the current density protection limit, constraint equation no. 35 should be turned on with iteration variable no. 53 (`fjprot`).

3.1.8 Poloidal field coils

The poloidal field (PF) coils are used initially to cancel the vertical field produced at the centre of the plasma by the central solenoid (Section 3.1.9) during start-up, and then to maintain the plasma position and shape during the flat-top period.

3.1.8.1 PF coil positions

The positions and sizes of the PF coils are partly input, and partly calculated after consideration of the required currents and allowable current density.

The PF coil locations are controlled using a set of switches stored in array `ipfloc` (see Figure 3.1), and are calculated in routine `PFCOIL`. The coils are (usually) organised into groups containing two PF coils placed symmetrically above and below the midplane, and each group `j` has an element `ipfloc(j)` assigned to it. Input parameter `ngrp` should be set to the number of groups, and `ncls(j)` should be assigned the number of coils in each group — which should be `2` in each case.

In the following, all variables are defined in the variable descriptor file `vardes.html`. The values for `rpf1`, `rpf2`, `zref(j)` and `routr` should be adjusted by the user to locate the PF coils accurately.

The three possible values of `ipfloc(j)` correspond to the following PF coil positions: (Redo taking into account snull and other recent changes e.g. rclsnorm)

`ipfloc(j) = 1 :` PF coils are placed above the central solenoid (one group only);

$$\begin{aligned} R &= \texttt{rohc} + \texttt{rpf1} \\ Z &= \pm(\texttt{hmax} * \texttt{ohhghf} + 0.1 + 0.5 * (\texttt{hmax} * (1.0D0 - \texttt{ohhghf}) + \texttt{tfcth} + 0.1)) \end{aligned}$$

`ipfloc(j) = 2 :` PF coils are placed above the TF coils (one group only);

$$\begin{aligned} R &= \texttt{rmajor} + \texttt{rpf2} * \texttt{triang} * \texttt{rminor} \\ Z &= \pm(\texttt{hmax} + \texttt{tfcth} + 0.86) \end{aligned}$$

`ipfloc(j) = 3 :` PF coils are placed radially outside the TF coils (any number of groups (?));

$$\begin{aligned} R &= \texttt{rtot} + \texttt{tfthko}/2.0D0 + \texttt{routr} \\ Z &= \pm(\texttt{rminor} * \texttt{zref(j)}) \end{aligned}$$

3.1.8.2 PF coil currents

The PF coil currents are calculated in routine `EFC`, and vary as a function of time during the tokamak operation as indicated in Figure 3.6.

3.1.8.3 PF coil resistance

The PF coils can be either resistive or superconducting. This is determined from the value of `ipfres`. If `ipfres = 0`, the PF coils and the central solenoid are assumed to be superconducting. If `ipfres = 1`, they are assumed to be resistive, with their resistivity given by the value of variable `pfclres`.

3.1.8.4 Superconducting materials

If `ipfres = 0`, switch `isumatpf` specifies which superconducting material is to be used for the PF coils and the central solenoid:

`isumatpf = 1 :` binary Nb_3Sn superconductor

`isumatpf = 2 :` ternary Nb_3Sn superconductor

`isumatpf = 3 :` NbTi superconductor

3.1.9 Ohmic heating coil

The ohmic heating (OH) coil is a PF coil used primarily during start-up (but also during the burn phase) to create and maintain the plasma current by inductive means. Swinging (changing) the current through the central solenoid causes a change in the flux linked to the plasma region, inducing a current in it. `PROCESS` calculates the amount of flux required to produce the plasma current, and also the

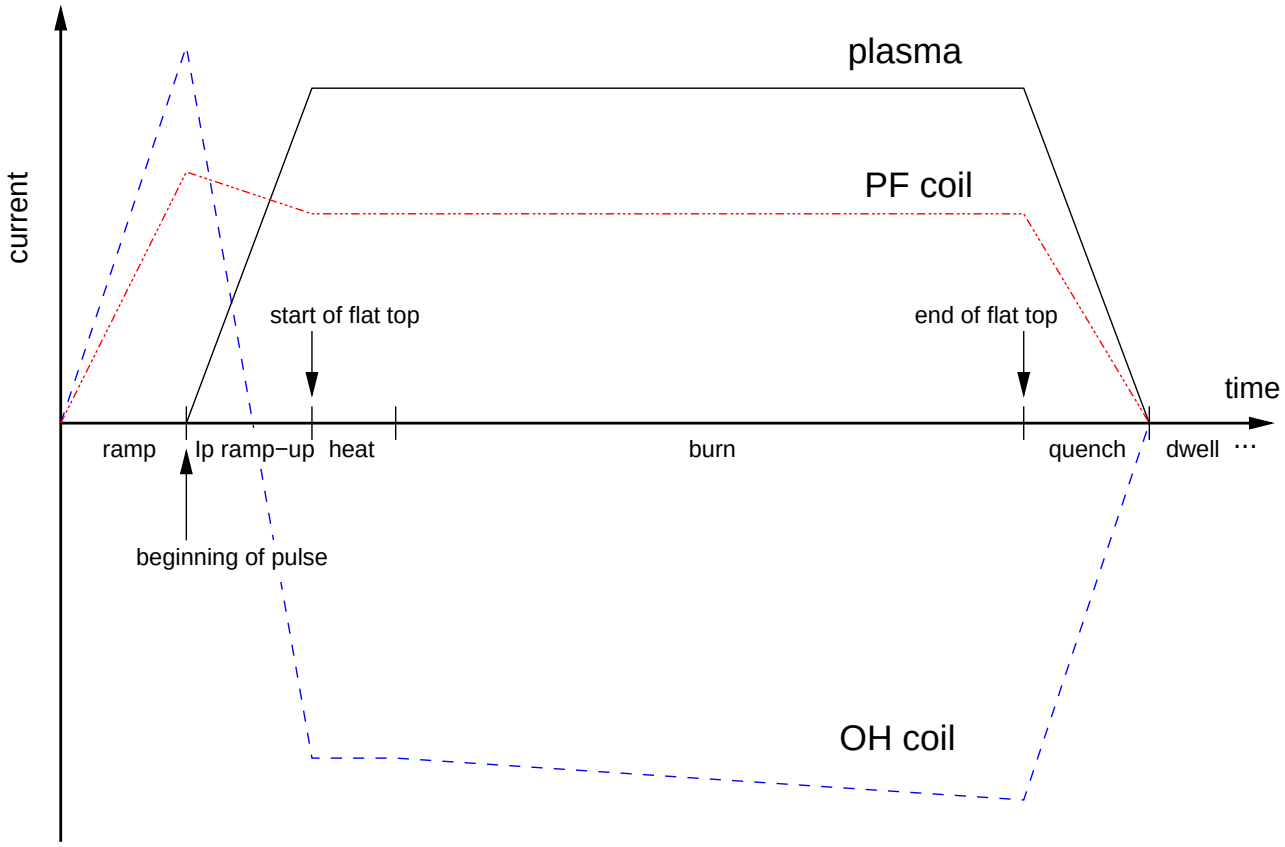

Figure 3.6: *Plot showing schematically the current waveforms for the plasma, a typical PF coil, and the central solenoid.*

amount actually available. The code measures the magnetic flux in units of Volt-seconds (= Webers). The central solenoid can be either resistive or superconducting (controlled via switches `ipfres` and `isumatpf` as for the other PF coils).

Switch `iohcl` controls whether a central solenoid is present. A value of `1` denotes that this coil is present, and should be assigned a non-zero thickness `ohcth`. A value of `iohcl = 0` denotes that no central solenoid is present, in which case the thickness `ohcth` should be set to zero. No PF coils should be located at positions defined by `ipfloc(j) = 1` if no central solenoid is present.

3.1.9.1 Plasma current ramp-up time

In the steady-state power plant scenario (`lpulse` $\neq$ `1` — see Section 3.3), the length of time taken for the central solenoid current to reverse (which is equal to the plasma current ramp-up time — see Figure 3.6) is determined from the value of switch `tohsin`. If `tohsin = 0.0D0`, then the plasma current ramp-up time `tohs` in seconds is given by `tohs` $= I_p/0.5$, where I_p is the plasma current in MA. Furthermore, the PF coil ramp time `tramp` and shutdown time `tqnch` are set equal to `tohs`. If `tohsin` $\neq$ `0.0D0`, the plasma current ramp-up time `tohs = tohsin`, and the PF coil ramp and shutdown times are input parameters.

If, however, a pulsed power plant is being modelled (`lpulse = 1`), the plasma current ramp-up time `tohs` is either an input parameter, or it can be iterated by using iteration variable 65. The ramp and shutdown times in the pulsed case are always set equal to `tohs`. To ensure that the plasma current ramp rate during start-up is prevented from being too high, as governed by the requirement to maintain plasma stability in l_i-q_ψ space, constraint equation no. 41 should be turned on with iteration variable no. 66 (`ftohs`).

3.1.9.2 Current density limits

The current density in the central solenoid can be limited at the beginning-of-pulse (BOP) and at the end-of-flat-top (EOF — see Figure 3.6). The limiting value is dependent on the maximum allowable stress in the coil as given by the value of `sigpfalw`. To limit the current density at the BOP, constraint equation no. 27 should be turned on with iteration variable no. 39 (`fjohc0`). To limit the current density at the EOF, constraint equation no. 26 should be turned on with iteration variable no. 38 (`fjohc`).

3.1.10 Auxiliary power systems: heating and current drive

3.1.10.1 Current Drive

The use of purely inductive current drive leads to pulsed plant operation because of the limited flux swing that can be achieved using the central solenoid. This poses problems due to the fact that fatigue failures may result, and there would also be a need for thermal storage to maintain a level supply between pulses. However, the plasma current can also be produced and maintained (partially or wholly) using non-inductive means which, in principle, removes this restriction. `PROCESS` contains a number of auxiliary current drive schemes, including various RF methods (Lower Hybrid, Electron Cyclotron, and Ion Cyclotron (Fast Wave) current drives) and also Neutral Beam current drive systems. The code calculates the efficiency and the resulting power requirements of the chosen system.

The fraction of the required plasma current to be produced by non-inductive means, `fvsbrnni`, should

be set, and flag `irfcd` should be set to `0` for purely inductive scenarios, or `1` otherwise. The current drive efficiency model to be used in this latter case is defined by the value of switch `iefrf`:-

`iefrf = 1 :` Fenstermacher Lower Hybrid model

`iefrf = 2 :` Ion cyclotron model [14]

`iefrf = 3 :` Fenstermacher electron cyclotron resonance model

`iefrf = 4 :` Ehst Lower Hybrid model

`iefrf = 5 :` ITER neutral beam model [14, 15]

`iefrf = 6 :` Culham Lower Hybrid model [15]

`iefrf = 7 :` Culham electron cyclotron model [15]

`iefrf = 8 :` Culham neutral beam model [15]

`iefrf = 9 :` Oscillating Field current drive (RFPs only — see Section 3.6.1.5)

(Note that, at present, the neutral beam models assume parabolic plasma profiles only.)

It is sometimes useful to adjust artificially the current drive efficiency values produced by these routines. This can be achieved by setting the scaling coefficient `feffcd`. The wall plug to plasma efficiencies can also be adjusted, by changing the relevant variable (`etaech`, `etalh`, `etanbi` or `etaof`).

3.1.10.2 Plasma heating

In addition to current drive, some auxiliary power can be used purely to heat the plasma. The value of input parameter `pheat` determines the amount of auxiliary *heating* power (in Watts) to be applied to the plasma. This variable may be used as an iteration variable (no. 11).

3.1.10.3 Neutral beam access

If present, a neutral beam injection system needs sufficient space between the TF coils to be able to intercept the plasma tangentially. The major radius `rtanbeam` at which the centreline of the beam is tangential to the toroidal direction is user-defined using input parameter `frbeam`, which is the ratio of `rtanbeam` to the plasma major radius `rmajor`. The maximum possible tangency radius `rtanmax` is determined by the geometry of the TF coils — see Figure 3.7, and this can be enforced using constraint equation no. 20 with iteration variable no. 33 (`fportsz`). The thickness of the beam duct walls may be set using input parameter `nbshield`.

3.1.10.4 Ignited plasma

Switch `ignite` can be used to denote whether the plasma is ignited, i.e. fully self-sustaining without the need for any injected auxiliary power during the burn. If `ignite = 1`, the calculated injected power does not contribute to the plasma power balance (Section 3.1.2.10), although the cost of the auxiliary power system is taken into account (the system is then assumed to be required to provide heating etc. during the plasma start-up phase only — use `pheat` to indicate the power requirement). If `ignite = 0`, the plasma is not ignited, and the auxiliary power is taken into account in the plasma power balance during the burn phase. Also, constraint equation no. 28 can be turned on to enforce the fusion gain Q to be at least `bigqmin`.

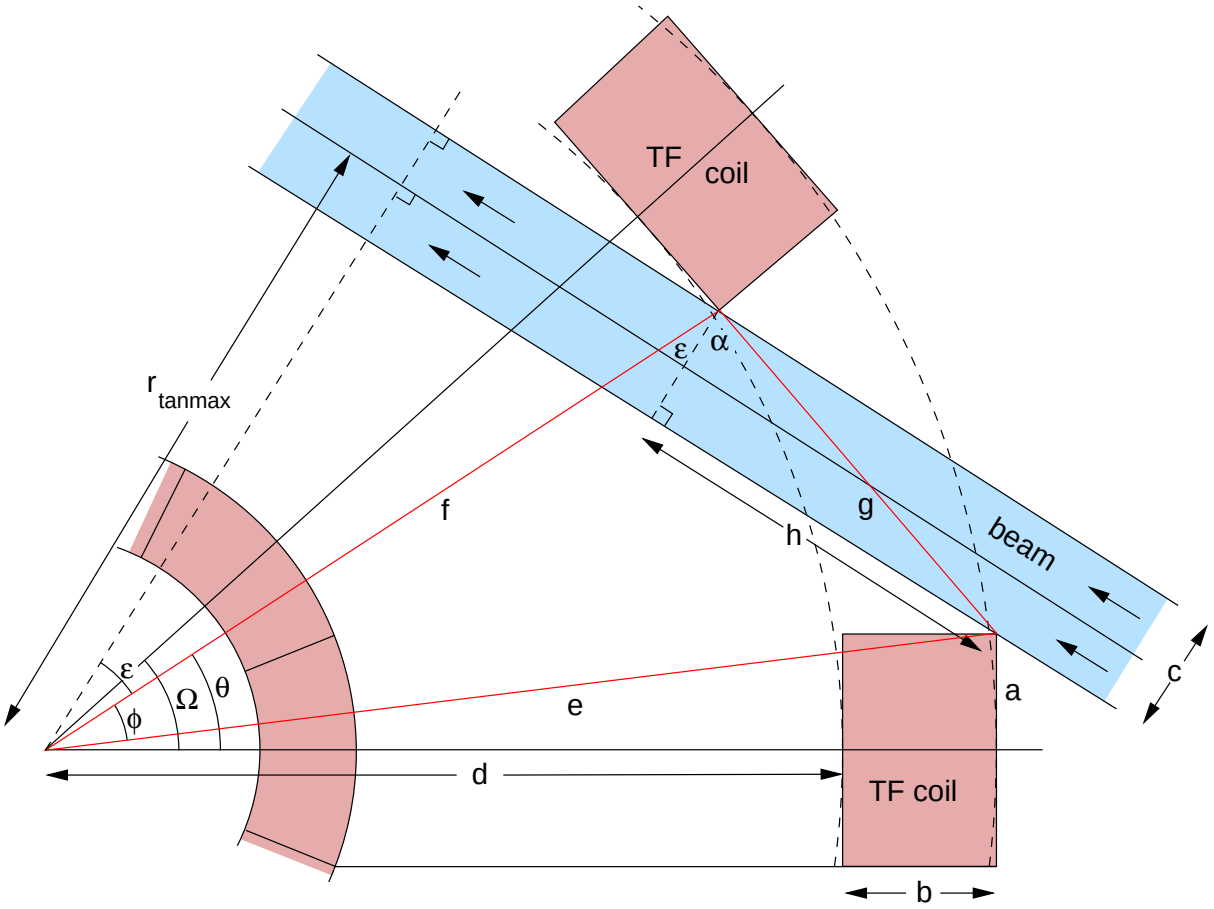

Figure 3.7: *Top-down schematic view of the neutral beam access geometry. The beam with the maximum possible tangency radius is shown here.*

3.1.11 Structural components

Structural components are required to provide support for the fusion power core systems against gravity and the magnetic forces that will be encountered during operation. The required structural masses and their costs are calculated.

3.1.12 Cryostat and vacuum system

The internal vacuum vessel provides a toroidal evacuated chamber containing the plasma, first wall, blanket and shield, and the space between this item and the external cylindrical cryostat encloses those components that need to operate at liquid helium temperatures. These include any superconducting (TF or PF) coils and the inter-coil structure. PROCESS calculates the cryogenic power load and the resulting heat exchanger requirements.

The vertical distance h between the uppermost PF coil and the external cryostat lid may be adjusted by changing the value of input parameter clhsf; a scaling based on ITER is used:

$$h = \texttt{clhsf} \left(\frac{2 \times \texttt{rdewex}}{28.440} \right) \tag{3.20}$$

The vacuum system is used for four different processes. Firstly, before plasma operations the chamber must be evacuated to remove outgassed impurities from the structure. Secondly, the chamber must be re-evacuated between burn operations. Thirdly, helium ash must be removed to prevent it from diluting the fuel. Finally, deuterium and tritium is removed on a steady state basis. PROCESS calculates the parameters of a vacuum system that satisfy all four requirements, with the option of either turbo pumps or cryo pumps being used.

Switch ntype controls whether a turbopump (ntype = 0) or a cryopump (ntype = 1) is used in the vacuum system.

3.1.13 Power conversion and heat dissipation systems

The PROCESS power plant takes into account all the systems required to perform the necessary conversion of fusion power to electricity, from the coolant systems in the plant components to the heat exchangers and turbines.

Figures 3.8 and 3.9 show schematically the overall power transfer mechanisms within the power plant outside of the plasma (the power flow *within* the plasma is shown in Figure 3.4).

(Note that these figures and the following description assume that switch ipowerflow is set to 1; its default value is currently zero, as the new power flow model is in draft form.)

Figure 3.8 shows how the power contributions originating from the plasma are converted to thermal power in the first wall, blanket, shield and divertor.

The radiation and neutron power contributions are distributed according to the following area fractions of a theoretical first wall with 100% coverage, which may be set as input parameters: fdiv (divertor area fraction), fhcd (heating / current drive / diagnostics apparatus), and fhole (any other gaps). The remaining fraction (1-fdiv-fhcd-fhole) is the area fraction of the actual first wall.

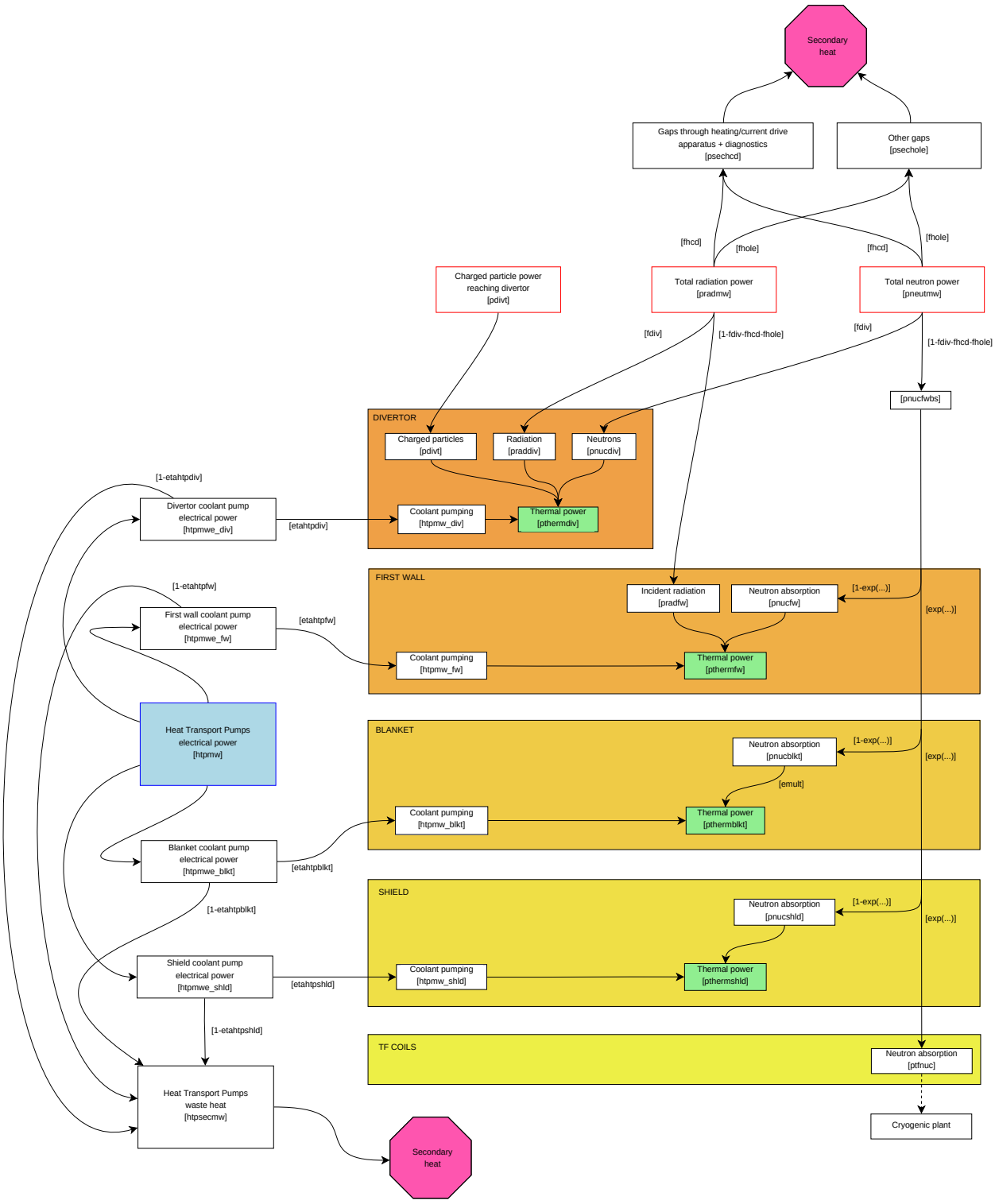

Figure 3.8: *Schematic diagram of the flow of power from the plasma to the thermal power deposited within the divertor, first wall, blanket and shield. Variable names are given in [. . .]. The three red-bordered contributions are passed from Figure 3.4, while the blue box comes from Figure 3.9.*

3.1.13.1 Divertor

All of the charged particle transport power leaving the plasma is assumed to be absorbed in the divertor, along with a proportion `fdiv` of the radiation power and the neutron power. The power necessary to drive the divertor coolant pumps P (= `htpmw_div` in Figure 3.8) is assumed to be a fraction f (= input parameter `fpumpdiv`) of the total thermal power absorbed by the divertor, which includes the coolant pump power itself:

$$\begin{aligned} P &= f.(\texttt{pdivt} + \texttt{praddiv} + \texttt{pnucdiv} + P) \\ \Longrightarrow P &= \frac{f}{1-f}.(\texttt{pdivt} + \texttt{praddiv} + \texttt{pnucdiv}) \end{aligned}$$

The efficiency of the divertor coolant pump is given by input parameter `etahtpdiv`.

Switch `iprimdiv` may be used to specify whether the thermal power deposited in the divertor becomes primary (high-grade) thermal power (`iprimdiv = 1`) or secondary (low-grade) thermal power (see Figure 3.9).

3.1.13.2 First wall

All of the remaining radiation power (i.e. the fraction not incident upon the divertor or lost through holes or the heating / current drive apparatus) is assumed to be absorbed by the first wall. The same fraction of the neutron power is assumed to pass into the first wall. Of this neutron power, a fraction $(1 - e^{\lambda/d})$ is assumed to be absorbed in the first wall (of thickness d), given by the neutron decay length λ in the first wall (input parameter `declfw`).

As for the divertor, the first wall coolant pump power is a fraction `fpumpfw` of the total thermal power absorbed by the first wall, and the efficiency of the first wall coolant pump is `etahtpfw`.

3.1.13.3 Blanket

A fraction (`1 - exp(declblkt/blnkoth)`) of the neutron power incident on the blanket is assumed to be absorbed in the blanket, where input parameter `declblkt` is the neutron decay length in the blanket. This absorbed neutron power is multiplied by the energy multiplication factor `emult` to calculate its contribution to the thermal power within the blanket.

As for the other components, the blanket coolant pump power is a fraction `fpumpblkt` of the total thermal power absorbed by the blanket, and the efficiency of the blanket coolant pump is `etahtpblkt`.

3.1.13.4 Shield

A fraction (`1 - exp(declshld/shldoth)`) of the neutron power incident on the shield is assumed to be absorbed in the shield, where input parameter `declshld` is the neutron decay length in the shield.

As for the other components, the shield coolant pump power is a fraction `fpumpshld` of the total thermal power absorbed by the shield, and the efficiency of the shield coolant pump is `etahtpshld`.

Switch `iprimshld` may be used to specify whether the thermal power deposited in the shield becomes primary (high-grade) thermal power (`iprimshld = 1`) or secondary (low-grade) thermal power (see Figure 3.9).

The remaining neutron power after passing through the shield is assumed to be absorbed in the TF coils as quantity `ptfnuc`. This power contributes to the total cryogenic power requirements of the machine.

3.1.13.5 Power conversion outside of the fusion power core

Figure 3.9 summarises the power conversion mechanisms outside of the fusion power core.

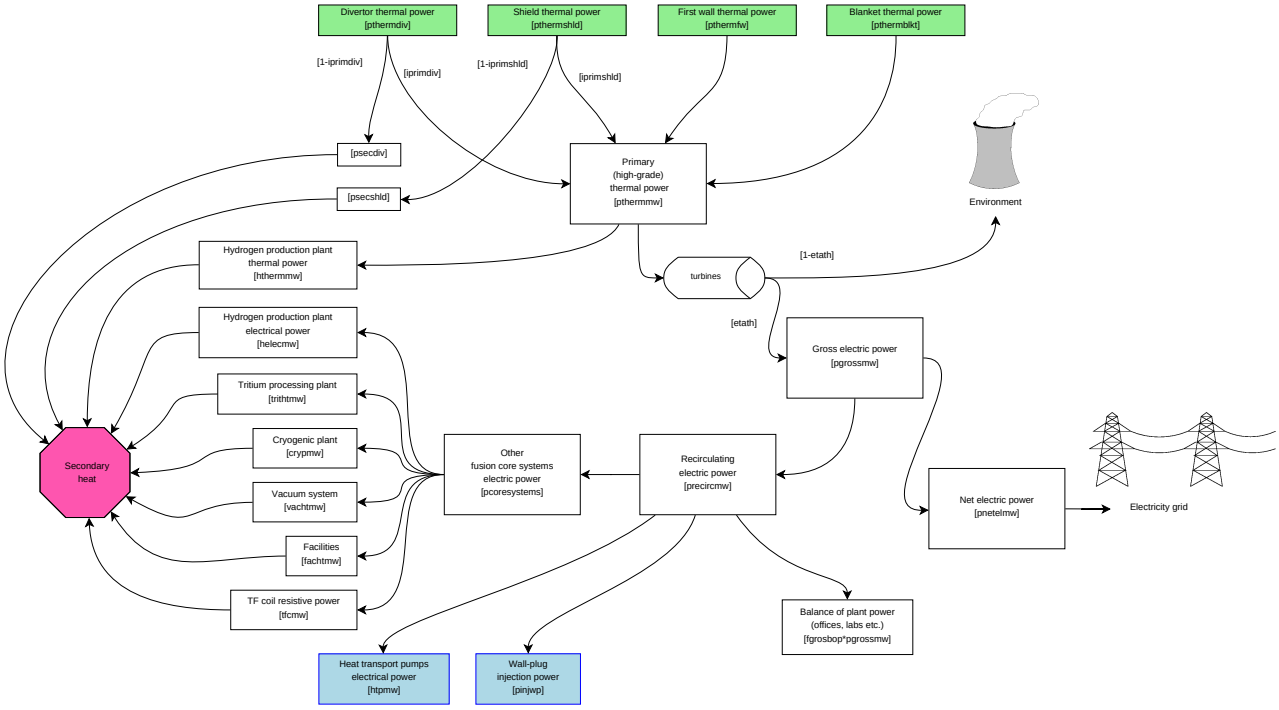

Figure 3.9: *Schematic diagram of the flow of power beyond the fusion power core, showing the origin of the primary and secondary thermal power and the electrical power balance within the plant. Variable names are given in [. . .]. The input power contributions in green boxes originate in Figure 3.8.*

The primary (high-grade, i.e. useable) thermal power comprises the thermal power contributions from the first wall and blanket, and possibly also those from the divertor and shield, depending on the values of the switches `iprimdiv` and `iprimshld` as described in the previous sections. The primary thermal power (less any thermal power required to produce hydrogen in a hydrogen production plant — see Section 3.4) is used to produce steam to turn the turbines, thus generating the plant's gross electrical power, with an efficiency given by input parameter `etath`. The power lost via the turbines' inefficiency escapes from the plant to the environment.

The electrical power required to operate the power plant itself is the so-called recirculating electric power. Any surplus is exported to the electricity grid as net electric power.

The recirculating power comprises the electrical power required to run all of the associated electrical systems surrounding the fusion power core, plus the on-site building services, offices, etc., as shown in Figure 3.9. Of these, the cryogenic plant power includes the power required to cool the TF coils from the neutron power absorbed by the coils, the PF coils (as defined by the ratio of the total PF coil stored energy to the fusion power pulse time `tpulse`), and other 'cold' components.

3.1.14 Buildings

The volume and ground area of all the various buildings on a power plant site are included in the `PROCESS` calculations for the benefit of the costing algorithms.

3.2 Spherical Tokamak Model

PROCESS has the ability to perform studies on tokamaks in the low aspect ratio regime (major radius $\leq 2\times$ minor radius). The physics and engineering issues [32] associated with these machines are somewhat different from those of conventional aspect ratio, and this is reflected by the following special models [2] in PROCESS.

1. The inboard build of a spherical tokamak (ST) is very different from that in a conventional tokamak. There is no inboard blanket (and possibly no inboard shield), and the inboard TF coil legs are replaced by a single centrepost. The radial build is altered so that, starting from the centreline ($R = 0$), the component order is: TF coil, gap, central solenoid, vacuum vessel, and then continuing as in Figure 3.1 (a D-shaped cross-section is assumed for the first wall, blanket, shield and vacuum vessel).

2. Spherical tokamaks have resistive TF coils that combine into a single centrepost at the centre of the machine. The centrepost is constructed from copper (as are the outboard TF coil sections), and is tapered lengthways so that it is narrowest at the midplane of the device. Routine CNTRPST calculates various parameters relevant to the centrepost, including the pump pressure, maximum temperature and pipe radius, and these may be limited using constraint equations 43 to 46 if required:

   - Equation 43 is a consistency equation for the average centrepost temperature.
   - Equation 44 can be used to limit the peak centrepost temperature to a maximum value (ptempalw) using iteration variable no. 68 (fptemp).
   - Equation 45 can be used to force a lower limit to the edge safety factor q_{lim} (see below), using iteration variable no. 71 (fq).
   - Equation 46 can be used to apply an upper limit to the ratio of plasma current to TF coil ("rod") current, using iteration variable no. 72 (fipir).

3. A gaseous divertor model is used, and a simple divertor heat load calculation is employed, rather than the more complex divertor model assumed for conventional aspect ratio tokamaks.

4. A simple PF coil current scaling algorithm is available for use with the ST option.

5. The plasma shaping terms (elongation and triangularity) can be calculated directly given the aspect ratio, using `ishape = 1` (see Section 3.1.2.1). This setting also scales the lower limit [2] for the edge safety factor, for use with constraint equation no. 45:

$$q_{lim} = 3\,(1 + 2.6 * \epsilon^{2.8}) \tag{3.21}$$

   where $\epsilon = a/R$.

6. Among the physics models that differ from those relevant to conventional aspect ratio machines are (i) the plasma poloidal field B_{pol}, (ii) the bootstrap current fraction, (iii) the beta limit, and (iv) the neutron heating of the centrepost [2].

3.2.1 Spherical tokamak switches

Switch itart provides overall control of the ST switches within the code, and subroutine CHECK ensures that no conflicting values are inadvertently set by the user in the input file. Table 3.4 summarises the switch values relevant to each aspect ratio regime.

switch	conventional aspect ratio `itart = 0`	low aspect ratio `itart = 1`
`ishape`	0, 2	0, 1
`ibss` (Section 3.1.2.11)	1, 2, 3	2, 3
`icurr` (Section 3.1.2.7)	1, 3, 4, 5, 6, 7	2
`itfsup` (Section 3.1.7)	0, 1	0

Table 3.4: *Summary of the switch values in PROCESS that relate to conventional aspect ratio and low aspect ratio machines.*

3.3 Pulsed Plant Operation

If the plasma current is not to be driven by purely non-inductive means, it is necessary to operate the plant in a pulsed manner as the current swing in the OH/PF coils cannot be continued indefinitely. `PROCESS` can perform a number of calculations relevant to a pulsed power plant, as detailed below.

Switch `lpulse` determines whether the power plant is assumed to be based on steady-state (`lpulse = 0`) or pulsed (`lpulse = 1`) operation.

3.3.1 Thermal cycling package

This performs calculations on the first wall of the machine. Evaluation of the mechanical and thermal stresses on this component lead to a measure of the maximum number of cycles to which the first wall can be subjected, and hence to the minimum allowable length of each reactor cycle for a specified first wall lifetime. The cycle time can be constrained to be at least the minimum value by turning on constraint equation no. 42 with iteration variable no. 67 (`ftcycl`).

The thickness of the first wall is constrained to lie within lower and upper bounds, which ensures that it can withstand the internal coolant pressure and the peak temperature and neutron fluence.

Switch `itcycl` activates the desired model for the first wall axial stress calculations. If `itcycl = 1` (the default), the wall is fully constrained axially, and no bending can occur. If `itcycl = 2`, there is no constraint on the axial motion, but no bending can occur. Finally, if `itcycl = 3`, again there is no axial constraint, and bending is allowed to occur.

3.3.2 First wall coolant temperature rise limit

The rise in temperature of the first wall coolant can be limited to be no more than the value of `dtmpmx` by turning on constraint equation no. 38 with iteration variable no. 62 (`fdtmp`).

3.3.3 First wall peak temperature limit

The maximum first wall temperature can be limited to be no more than the value of variable `tpkmax` by turning on constraint equation no. 39 with iteration variable no. 63 (`ftpeak`).

3.3.4 Start-up power requirements

The minimum auxiliary power required during the start-up (ignition) phase is calculated on the basis of a POPCON analysis. Ignition is accessed via the so-called Cordey Pass (the path in plasma

density–temperature space which minimises the power requirement) and the code ensures that there is sufficient auxiliary power to accommodate this. In fact, this calculation is very CPU-intensive, so the relevant routine is not called at present. In practice, the auxiliary power tends to exceed the minimum allowable value anyway, without any need to constrain it to do so.

The auxiliary power reaching the plasma can be forced to be more than the minimum allowable value `auxmin` by turning on constraint equation no. 40 with iteration variable no. 64 (`fauxmn`). The value of `auxmin` is determined by the code if the start-up model is activated, otherwise it may be initialised via the input file.

3.3.5 Plasma current ramp-up time

This calculation ensures that the plasma current ramp rate during start-up is prevented from being too high, as governed by the requirement to maintain plasma stability in l_i - q_ψ space (see Section 3.1.9.1).

3.3.6 Burn time

The length of the burn time is calculated from the surplus volt-seconds available from the OH/PF coil system during the plasma burn phase, after the flux required during the plasma start-up is taken into account. A minimum burn time can be enforced via constraint equation no. 13 and iteration variable no. 21 (`ftburn`).

3.3.7 Thermal storage

During every cycle there is a period when no fusion power is produced. The net electric output from the plant must, however, be maintained, and this is achieved using thermal storage. There are three types of thermal storage available within `PROCESS`, and the value of switch `istore` determines which is to be used. If `istore = 1` (the default), option 1 of Ref. [33] is assumed, which utilises the thermal storage inherent in the machine's steam cycle equipment. This should be used if the machine down time is less than 100 seconds. If `istore = 2`, option 2 of Ref. [33] is assumed, which uses the same method as before, but augments it with an additional boiler. This may be used for machine down times of up to 300 seconds. Finally, if `istore = 3`, a large stainless steel block acts as the thermal storage medium.

3.4 Hydrogen Production Facility

Fusion power plants have been mooted as a means of producing hydrogen for use in fuel cells for cars, for instance. `PROCESS` includes options to enable the plant to produce hydrogen using a number of different processes.

To include the production of hydrogen by the power plant, it is necessary to set the switch `ihplant`, as follows:

`ihplant = 0 :` No hydrogen production (default)

`ihplant = 1 :` Hydrogen production by low temperature electrolysis

`ihplant = 2 :` Hydrogen production by endothermic high temperature electrolysis

`ihplant = 3 :` Hydrogen production by exothermic high temperature electrolysis

`ihplant = 4 :` Hydrogen production by thermo-chemical processes

Table 3.5 describes the additional options available for each of the types of hydrogen production given above. The different processes use either electrical power or thermal power directly, so the required inputs differ. Variable `helecmw` (iteration variable no. 87) is the electrical power in MW required for hydrogen production, while `hthermmw` (iteration variable no. 88) is the thermal power required. Note that `hthermmw` must not be used as an iteration variable if `ihplant` $\neq$ `4`, as it will be calculated from the required electrical power instead. Similarly, `helecmw` must not be used as an iteration variable if `ihplant = 4`. The efficiency variables given in Table 3.5 are all input parameters, and are the factors

hydrogen plant option	`helecmw`	`hthermmw`	efficiency variable
`ihplant=1`	input	zero	`etahlte`
`ihplant=2`	input	calculated	`etahten`
`ihplant=3`	input	calculated	`etahtex`
`ihplant=4`	zero	input	`etahth`

Table 3.5: *Summary of the variables in PROCESS that relate to the different hydrogen plant processes.*

to be used to convert the value of `helecmw` to the amount of hydrogen produced (in MW equivalent); these can be greater than unity in all cases except `ihplant = 4`.

3.5 Stellarator Model

The code has the ability to perform calculations based on the physics and engineering of a stellarator, which, although being a toroidal device, is radically different in a number of ways from a tokamak.

The model is largely based on W7-X and the HELIAS 5-B stellarator power plant design [34] (Figure 3.10) and related modelling that has been performed by IPP Greifswald [35, 36, 37].

To activate the stellarator coding, it is necessary to create a file `device.dat`, containing the single character `1` in the first row, in the working directory (see Section 4.1). This has the effect of setting the internally-used switch `istell = 1`. If the file is absent, or its first character is set to something other than `1`, the stellarator model is not used, and `istell` is set to `0`.

The stellarator model is largely contained within source file `stellarator.f90`. The following consistency equations (see Section 2.4) should be used without modification:

`1 :` plasma beta consistency

`2 :` global power balance

`11 :` radial build consistency

3.5.1 Stellarator physics

Much of the physics is identical to that for tokamaks, including the plasma composition and the fusion power calculations. However, some physics topics do differ between stellarators and tokamaks, as follows.

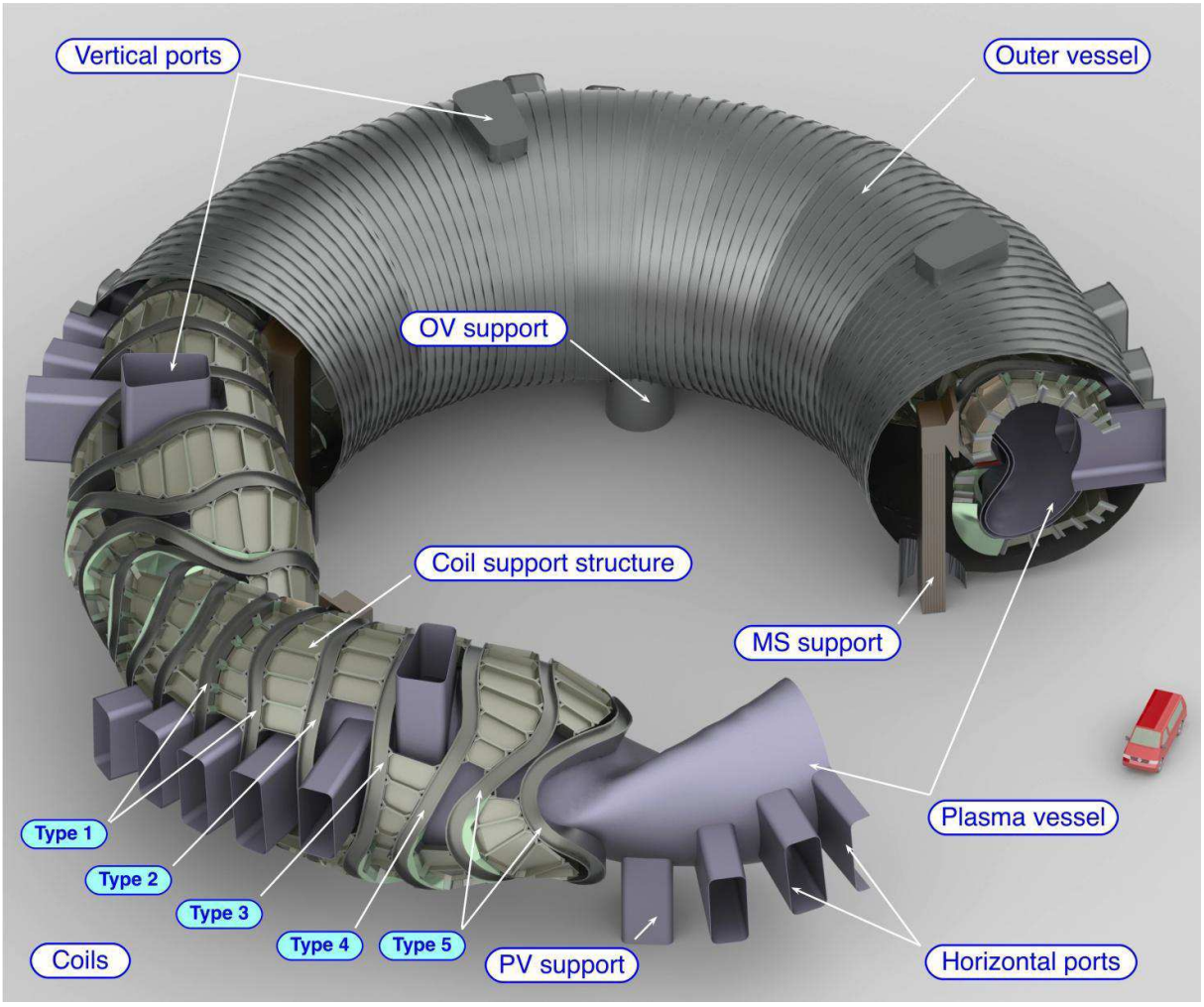

Figure 3.10: *Fusion power core of the HELIAS 5-B conceptual power plant design (from [34])*

3.5.1.1 Plasma geometry

The plasma geometry model uses Fourier coefficients to represent the complex 3-D plasma shape typical of stellarators. A VMEC [38] calculation (or other equilibrium code that can provide the Fourier coefficients of the LCFS) must have been performed prior to running with `PROCESS`. The overall size of the plasma is scaled correctly for the required (mean) major and minor radii for the machine being modelled [39].

It is necessary to provide three input files that define the plasma shape:

1. Set `vmec_info_file` in the input file to the name of a file with the following format (the numbers given are those for the W7-X high mirror configuration); in practice only the effective major and minor radius values (`R_eff` and `a_eff`, respectively) and the number of field periods are used:

```
R_eff [m]   a_eff [m]   Aspect ratio   Plasma vol [m3]  LCFS surf area [m2]  field periods
5.486576    0.4942638   11.10050       26.45749         133.4281             5.0
```

2. Set `vmec_rmn_file` in the input file to the name of a VMEC output file containing the calculated plasma boundary $R(m, n)$ Fourier coefficients, where $m = 0$ to 11 (rows) and $n = -12$ to 12 (columns).

3. Set `vmec_zmn_file` in the input file to the name of a VMEC output file containing the calculated plasma boundary $Z(m, n)$ Fourier coefficients, where $m = 0$ to 11 (rows) and $n = -12$ to 12 (columns).

This method enables a wide range of potential plasma geometries to be modelled, if required. The plasma volume, surface area and mean cross-sectional area are the outputs from the plasma geometry model.

3.5.1.2 Absense of plasma current

Stellarators try to achieve zero plasma current in order to allow safe divertor operation, so no current scalings are required.

3.5.1.3 Beta limit

The beta limit is assumed to be 5%, based on 3-D MHD calculations [40]. To apply the beta limit, constraint equation no. 24 should be turned on with iteration variable no. 36 (`fbetatry`).

3.5.1.4 Density limit

The density limit relevant to stellarators has been proposed to be [41]

$$n_{\max} = 0.25 \left(P B_0 / R_0 \, a_p^2 \right)^{\frac{1}{2}} \tag{3.22}$$

where n is the line-averaged electron density in units of 10^{20} m^{-3}, P is the absorbed heating power (MW), B_0 is the on-axis field (T), R_0 is the major radius (m), and a_p is the plasma minor radius (m). To enforce the density limit, turn on constraint equation no. 5 with iteration variable no. 9 (`fdene`).

3.5.1.5 τ_E scaling laws

Five confinement time scaling laws relevant to stellarators are present within PROCESS. The value of switch isc determines which of these is used in the plasma energy balance calculation.

$$\tau_E \text{ (Large Helical Device [41]: isc=21)} = 0.17\, R_0^{0.75}\, a_p^2\, \bar{n}_{20}^{0.69}\, B_0^{0.84}\, P^{-0.58} \quad (3.23)$$

$$\tau_E \text{ (Gyro-reduced Bohm [42]: isc=22)} = 0.25\, B_0^{0.8}\, \bar{n}_{20}^{0.6}\, P^{-0.6}\, a_p^{2.4}\, R_0^{0.6} \quad (3.24)$$

$$\tau_E \text{ (Lackner-Gottardi [43]: isc=23)} = 0.17\, R_0\, a_p^2\, \bar{n}_{20}^{0.6}\, B_0^{0.8}\, P^{-0.6}\, \iota^{0.4} \quad (3.25)$$

$$\tau_E \text{ (ISS95 [44]: isc=37)} = 0.079\, R_0^{0.65}\, a_p^{2.21}\, \bar{n}_{19}^{0.51}\, B_0^{0.83}\, P^{-0.59}\, \bar{\iota}^{0.4} \quad (3.26)$$

$$\tau_E \text{ (ISS04 [45]: isc=38)} = 0.134\, R_0^{0.64}\, a_p^{2.28}\, \bar{n}_{19}^{0.54}\, B_0^{0.84}\, P^{-0.61}\, \bar{\iota}^{0.41} \quad (3.27)$$

Here, $\bar{\iota}$ is the rotational transform, which is equivalent to the reciprocal of the tokamak safety factor q.

3.5.1.6 Heating power options

Stellarators require no current drive, although provision for auxiliary heating does need to be present. The method by which auxiliary heating power is supplied is determined by the switch isthtr:

isthtr = 1 : electron cyclotron resonance heating

isthtr = 2 : lower hybrid heating

isthtr = 3 : neutral beam injection

The value of variable pheat determines the actual amount of auxiliary heating power (in Watts) to be applied to the plasma. This variable may be used as an iteration variable (no. 11). Switch ignite may be used if necessary — see Section 3.1.10.4.

3.5.1.7 Divertor

Although the divertor has the same function in both stellarators and tokamaks, the envisaged concepts differ quite substantially. This is not only related to the different geometry and symmetries but also specifically to the magnetic configuration. While the inherent axisymmetry of a tokamak allows for poloidally-continuous single or double divertor configurations, the periodicity of helical advanced stellarators leads to multiple X-points with a corresponding chain of magnetic islands. This island structure may be exploited by carefully placing divertor plates along the stochastic field lines, naturally leading to a discontinuous divertor, with the individual plates being placed in a complex 3-D geometry; see Figure 3.11.

Rather than trying to describe the complex physics with a two-point scrape-off layer model as is used for tokamaks, the stellarator divertor model [36] is based on fundamental principles which relate the power crossing the separatrix with an effective wetted area on the divertor plates allowing the code to estimate the heat load delivered to the divertor. A basic island divertor model is used which assumes diffusive cross-field transport and high radiation at the X-point,

The radiated power fraction in the scrape-off layer is given by the input parameter f_rad. This is in contrast to the method used for tokamaks, in which the radiated power fraction is a calculated quantity.

A number of other input parameters may be used to modify the divertor model; see the variable descriptor file for more details.

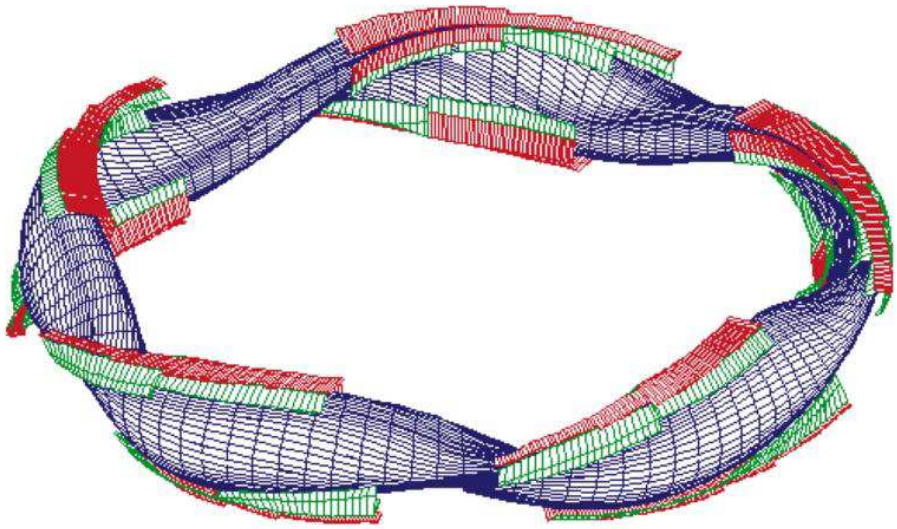

Figure 3.11: *A five-period HELIAS plasma (specifically W7-X) with island divertor plates shown in red*

3.5.2 Machine configuration

There are a large number of possible stellarator configurations. As stated earlier, the one chosen for the PROCESS model is based on the **HELI**cal **A**dvanced **S**tellarator (HELIAS) concept, in which all the coils resemble distorted, non-planar TF coils — no helical coils or tokamak-like PF coils are present. The coil geometry is scaled from the HELIAS 5-B power plant design, which is based on a five field-period HELIAS configuration.

3.5.2.1 Machine build

Since a stellarator is inherently non-axisymmetric, the build of the PROCESS stellarator is defined in terms of the mean thicknesses of components.

The surface areas for the first wall, blanket and shield components are scaled linearly with their effective minor radius from the plasma surface area calculation (the area of a simple torus of circular cross-section is $4\pi^2 Ra$, hence the linear scaling with a). The input parameter `fhole` may be used to specify the hole fraction, to adjust the surface area to take account of ports and divertors, for instance. The volumes of the first wall etc. are simply given by the product of their surface area and their mean thickness.

In contrast to tokamaks, which in PROCESS are assumed to have a cylindrical external cryostat completely surrounding the fusion power core, the stellarator model assumes that the external cryostat (labelled as the outer vessel in Figure 3.10) is toroidal with a circular cross-section. Its cross-section is assumed to be centred at the mean plasma major radius.

All items external to the fusion power core (buildings, turbines, power conversion systems, etc.) remain unchanged.

3.5.2.2 Stellarator coils

The stellarator coil model [37, 46] combines scaling aspects based on the HELIAS 5-B design in combination with analytic inductance and field calculations.

The fully three-dimensional shape of the coils is assumed to be fixed, but the sizes of the coils are scaled from the HELIAS 5-B values to the geometrical values for the machine being modelled using fundamental physics and engineering principles.

The stellarator coils are assumed to be superconducting — no resistive coil calculations are performed. The critical field at the superconductor is calculated using circular approximations for the coils in the inductance and field calculations, and the limit is enforced automatically. Available superconductors are Nb_3Sn (`isumattf = 1`) and NbTi (`isumattf = 3`).

The winding pack cross-section is rectangular for the stellarator coils, rather than the complicated two-step cross-section assumed for tokamaks. The coil thicknesses and most of the dimensions of the materials within the coil cross-section are outputs from the model, instead of being inputs as is the case for tokamaks; see the variable descriptor file for details. In addition, certain iteration variables (`tfcth`, no. 13; `thkcas`, no. 57; `cpttf`, no. 60 and `tftort`, no. 77) should not be turned on in the input file as they are calculated self-consistently; the code will stop with an error message if this is attempted.

3.6 Reversed Field Pinch Model

In addition to the tokamak and stellarator magnetic confinement devices, the code has the ability to perform calculations based on the physics and engineering of a reversed field pinch (RFP) device. This third type of toroidal magnetic device is superficially similar in design to a tokamak (so therefore shares many of the same components), but the magnetic field configuration differs.

The model used in `PROCESS` is largely based on the TITAN fusion power plant [47, 48]. The following sections summarise its main features, where they differ from those for tokamaks.

To activate the RFP coding, it is necessary to create a file `device.dat`, containing the single character `2` in the first row, in the working directory (see Section 4.1). This has the effect of setting the internally-used switch `irfp = 1`. If the file is absent, or its first character is set to something other than `2`, the RFP model is not used, and `irfp` is set to `0`.

3.6.1 RFP physics

The plasma in RFPs is circular in cross-section and is axisymmetric.

3.6.1.1 Beta limit

The poloidal beta is limited to a maximum value given by input parameter `betpmx` (default = 0.19) using constraint equation 48 and iteration variable 79 (`fbetap`).

3.6.1.2 Density limit

No density limit is explicitly coded for RFPs, other than by simply constraining the upper bound of the electron density variable `dene` (iteration variable 6).

3.6.1.3 τ_E scaling law

One confinement time scaling law relevant to RFPs is present within `PROCESS`. The value of switch `isc` determines the scaling to be used in the plasma energy balance calculation.

$$\tau_E \text{ (TITAN RFP [47]: } \texttt{isc=25}) = 0.05\, a^2\, I_p(\text{MA})$$

3.6.1.4 F-Θ plot

Much of the RFP physics is derived from the characteristic F-Θ plot, where $F = B_\phi(a)/\langle B_\phi \rangle$, $\Theta = B_\theta(a)/\langle B_\phi \rangle$, and $\langle B_\phi \rangle$ is the average value of the toroidal field over the plasma cross-section. Given a value of the pinch parameter Θ, the corresponding value of the reversal parameter F may be read from the F-Θ plot, and from these the plasma current and the current in the TF coils may be obtained using

$$\begin{aligned} B_\theta(a) &= \frac{\mu_0 I_p}{2\pi a} \\ B_\phi(a) &= \frac{\mu_0 I_{TFC}}{2\pi R} \end{aligned}$$

(the second of these assumes that $B_\phi(a)$ is approximately equal to the vacuum toroidal field at the plasma centre).

The pinch parameter Θ is set using iteration variable no. 78: `rfpth`. The corresponding value of the reversal parameter F (`rfpf`) is calculated using routine `FTHETA`. F is constrained to be negative using constraint equation no. 49 with f-value `frfpf` (iteration variable no. 80).

3.6.1.5 Current drive

The RFP oscillating field current drive option is turned on by setting `iefrf = 9`, with a fixed efficiency of 0.8 Amps per Watt of power injected into the plasma (the coding for this model is, therefore, trivial). The wall plug to injector efficiency is set using input parameter `etaof`, which has a default value of 0.5, and the unit cost is set using input parameter `ucof`. The default value for `ucof` is 3.3 $ per Watt of injected power.

The bootstrap fraction is assumed to be zero.

Plasma ignition, and additional plasma heating using auxiliary power, are treated as for tokamaks.

3.6.2 TF coils

The TF coils for the RFP option are derived from the TITAN-II coil set, which uses circular copper coils. The radial thickness is set using `tfcth` as usual, but the toroidal thickness may also be set, using iteration variable no. 77 (`tftort`). This is constrained to be no larger than is geometrically possible using constraint equation no. 47 with iteration variable no. 76 (`frfptf`).

3.6.3 Ohmic heating coils

The TITAN-I ohmic heating coil locations, currents etc. are assumed. These comprise 14 copper coils, with currents swinging from positive to negative during the plasma start-up period, and then decaying back to zero. To account for different plasma geometries and currents, the ohmic heating coil locations relative to the plasma centre scale with the TF coil radius and with the plasma major radius, and the current per turn scales linearly with the plasma current.

3.6.4 EF coils

The Equilibrium Field coils are based on the TITAN-I EF coils, which are two superconducting (NbTi) coils that provide the correct vertical field at the plasma centre. The coil locations scale with the plasma major radius and TF coil radius along with the ohmic heating coils.

3.6.5 Divertor

The TITAN divertor concept uses three poloidal divertors, which bear little resemblance to the typical tokamak toroidal divertor systems. The code uses the TITAN divertor lifetime of one year to enable the divertor costs to be reasonable, although the divertor surface area (and therefore the cost per divertor) is likely to be inaccurate.

3.6.6 Code modifications

As with the stellarator model, the RFP model has been incorporated in such a way as to allow its simple removal again if required in the future. All new routines are confined to the dedicated source file `rfp.f90`. The tokamak-relevant consistency equations in `PROCESS` (see Section 2.4) are used without modification, to ensure that the coil currents and the fields they produce are consistent with the plasma parameters.

3.7 Inertial Fusion Energy Model

As well as magnetic confinement devices, `PROCESS` has the ability to model inertial fusion plants, in which a laser or ion beam is used to ignite a target pellet containing the fusion fuel.

To activate the inertial fusion energy (IFE) coding, it is necessary to create a file `device.dat`, containing the single character `3` in the first row, in the working directory (see Section 4.1). This has the effect of setting the internally-used switch `ife = 1`. If the file is absent, or its first character is set to something other than `3`, the IFE model is not used, and `ife` is set to `0`.

The IFE model [49] is controlled using two additional switches.

`ifetyp = 0 :` Generic device build

`ifetyp = 1 :` OSIRIS-type device build [50, 51, 52]

`ifetyp = 2 :` SOMBRERO-type device build [53, 54]

`ifetyp = 3 :` HYLIFE-II-type device build [55, 56, 57]

Switch `ifetyp` defines the type of device that is assumed; this varies widely between different conceptual designs. The generic type assumes a cylindrically symmetric device, while the other types are approximations to the builds of the given conceptual machines [58]. In general, the build from the centre of the device (at the target ignition location) is in the order: chamber, first wall, gap, blanket, gap, shield, gap, building wall. The user specifies the thicknesses of these regions, and also the materials that are present and in what proportions. [expand]

`ifedrv = -1 :` Driver efficiency and target gain are input as functions of driver energy

`ifedrv = 0 :` Driver efficiency and target gain are input

`ifedrv = 1 :` SOMBRERO laser drive efficiency and target gain assumed

`ifedrv = 2 :` OSIRIS heavy ion beam driver efficiency and target gain are assumed [59]

Switch `ifedrv` defines how the code calculates the driver efficiency and target gain — these are the primary outputs required from the physics part of the model. For the SOMBRERO and OSIRIS cases (`ifedrv = 1` and `ifedrv = 2`, respectively) the driver efficiency and gain are calculated from curves of these parameters as functions of the driver energy. For the `ifedrv = -1` case, the user provides the driver efficiency and target gain versus driver energy, via the two arrays `etave(1:10)` and `gainve(1:10)` respectively; the element number corresponds to the driver energy in MJ, and outside the range 1–10 MJ the curves are extrapolated linearly. Finally, for the `ifedrv = 0` case, the user inputs single values for the driver efficiency (`drveff`) and target gain (`tgain`).

Constraint equation no. 50 can be turned on to enable the ignition repetition rate to remain below a user-specified upper limit (`rrmax`); iteration variable no. 86 (`frrmax`) is the associated f-value (see Section 2.4). The other iteration variables relevant for the IFE model are nos. 81–85 (`edrive`, `drveff`, `tgain`, `chrad` and `pdrive` — see Table 2.3).

3.8 Safety and Environment Models

At present, the neutronics, activation and inventory calculations comprise the safety and environment models in the code.

The models comprising the safety and environmental calculations [60] within the code are all called from routine `FISPAC`. They are only performed once, at the end of each run, as they take a relatively long time to evaluate, and the results are only used for diagnostic purposes — no constraints are imposed at present to minimise doses, for instance.

N.B. These models are currently not available in the present version of the code.

3.8.1 Neutronics

The neutronics module predicts the neutron flux spectra in the inboard and outboard first wall and blanket components. The spectra are based on a simplified tokamak device that has a fixed ratio (= 1.5825) between the outboard blanket thickness and the inboard blanket thickness, and are scaled according to the actual thickness of the outboard blanket. This relatively limited, single-parameter approach is expected to be replaced by a more general method, which should allow a more accurate portrayal of the device being modelled by `PROCESS`.

3.8.2 Activation and inventory information

The code evaluates the consequences of exposing the power plant's materials to the calculated neutron fluxes, subject to the limitations imposed by the neutronics module. A library of neutron cross-sections and decay data is used to calculate the total activity, gamma-ray dose rate and decay heat output due to the materials' exposure to neutrons, both at the end of the plant's life and at a time 100 years later. These values are relevant to decommissioning and disposal studies, and additional parameters that can be obtained from the nuclide inventory will also be included as the need arises.

3.9 Cost Models

The cost accounting used by `PROCESS` combines methods [61] used in the TETRA code [1] and the Generomak [62] scheme. The costs are split into the standard accounting categories [63] generally used in the reporting of power plant costs. The best references for the algorithms used are [2], and source file `costs.f90` in the code itself.

The majority of the costed items have a unit cost associated with them. These values scale with (for example) power output, volume, component mass etc., and many are available to be changed via the input file. All costs and their algorithms correspond to 1990 dollars.

3.9.1 Cost options

3.9.1.1 N^{th} of a kind costs

The unit costs of the components of the fusion power core are relevant to "first-of-a-kind" items. That is to say, the items are assumed to be relatively expensive to build as they are effectively prototypes and specialised tools and machines have perhaps been made specially to create them. However, if a "production line" has been set up, and R & D progress has allowed more experience to be gained in constructing the power core components, the costs will be reduced as a result. Variable `fkind` may be used to multiply the raw unit costs of the fusion power core items (by a factor less than one) to simulate this cost reduction for an N^{th}-of-a-kind device. In other systems studies of fusion power plants [64], values for this multiplier have ranged from 0.5 to 0.8.

3.9.1.2 Level of safety assurance

Many of the unit costs have four possible choices, relating to the level of safety assurance [65] flag `lsa`. A value `lsa = 1` corresponds to a plant with a full safety credit (i.e. is truly passively safe). Levels `2` and `3` lie between the two extremes, and level `4` corresponds to a present day fission reactor, with no safety credit.

3.9.1.3 Replaceable components

The first wall, blanket, divertor, centrepost (if present) and current drive system have relatively short lifetimes because of their hostile environment, after which they must be replaced. Because of this frequent renewal they can be regarded as though they are "fuel" items, and can be costed accordingly. Switch `ifueltyp` is used to control whether this option is used in the code. If `ifueltyp = 1`, the costs of the first wall, blanket, divertor and a fraction `fcdfuel` of the cost of the current drive system are treated as fuel costs. If `ifueltyp = 0`, these are treated as capital costs.

3.9.1.4 Cost of electricity calculations

Switch `ireactor` determines the type of cost of electricity calculation that is performed. If `ireactor = 0`, no cost of electricity calculation is performed. If `ireactor = 1`, then the cost of electricity is evaluated, with the value quoted in units of m$/kWh.

3.9.1.5 Net electric power calculation

Related to the cost of electricity is the net electric power calculation performed in routine `POWER`. It is possible that the net electric power can become negative due to a high recirculating power. Switch `ipnet` determines whether the net electric power is scaled to always remain positive (`ipnet = 0`), or whether it is allowed to become negative (`ipnet = 1`), in which case no cost of electricity calculation is performed.

3.10 Other Switches and Models

3.10.1 Diagnostic output

Switch `verbose` can be set to `1` in the input file to allow the code to write some additional diagnostic output to the screen. Currently, extra information about the progress of the optimiser is produced. This setting is likely to be of interest only to "expert" users.

3.10.2 Code parameters affecting other models

This chapter has summarised the methods by which several of the models in the code can be activated. There are many others present, however, and it is suggested that the user refers to the variable descriptor file, `vardes.html`. As stated earlier in Section 2.2, this contains details of all the parameters within the code that can be changed by the user, in order to customise the machine modelled by `PROCESS`.

Chapter 4

Execution of the Code

The intention of this chapter is to provide a comprehensive prescription for setting up and performing runs with the code. Firstly, the input file's structure and format is described. The user is then taken through the procedure for setting up the code to model a new machine, and finally an attempt is made to indicate and solve the problems that the user will face whilst trying to achieve a feasible solution.

Please note: There is now a prototype Graphical User Interface (GUI) for people to use to run `PROCESS`; see Section 6.3 for details.

4.1 The Input File

The input file `IN.DAT` is used to change the values of the physics, engineering and other code parameters from their default values, and to set up the numerics (constraint equations, iteration variables etc.) required to define the problem to be solved.

4.1.1 Tokamak, stellarator, RFP or IFE?

In addition to the main input file `IN.DAT`, a second input file `device.dat` is used to signal to the code whether a tokamak, stellarator, reversed field pinch or inertial fusion enery plant is to be modelled. (If the file does not exist in the working directory, the standard tokamak model is used.)

File `device.dat` should contain a single character in the first line, which is interpreted as follows:

`0` : use tokamak model
`1` : use stellarator model
`2` : use reversed field pinch model
`3` : use inertial fusion energy model

4.1.2 File structure

The input file comprises a number of lines reminiscent of the Fortran `NAMELIST` format used by some programs to read in data. Except for comment lines that start with a `*` character, each line is of the form

```
variable = value
```

where `variable` is the name of one of the input parameters or iteration variables listed in the variable descriptor file, and `value` is the (usually numerical) initial value required for that variable. (Arrays, as opposed to scalar quantities, are treated differently — see below.) Variables can be specified in any order in the input file.

The routines that read in and parse the data from the input file are in source file `input.f90`. They allow a great deal of error-trapping to be carried out at the input stage. All input data are screened for non-sensible values directly; this is a useful feature of the code, since without such intervention modern computers are notorious at not terminating programs when an arithmetically impossible or undefined operation ("`NaN`" error) is encountered.

4.1.3 Format rules

The following rules must be obeyed when writing an input file:

1. Each variable must be on a separate line.
2. Variable names can be upper case, lower case, or a mixture of both.
3. Spaces may not appear within a variable name or data value.
4. Other spaces within a line, and trailing spaces, are ignored.
5. Commas are not necessary between variables (but see below).
6. Data can extend over more than one line.
7. One-dimensional arrays can be explicitly subscripted, or unscripted, in which case the following element order is assumed: `A(1), A(2), A(3),...`
8. At present, multiple dimension arrays can only be handled without reference to explicit subscripts, in which case the following element order is assumed: `B(1,1), B(2,1), B(3,1),...` The use of the input file to specify multiple dimension array elements is prone to error.
9. Unscripted array elements must be separated by commas.
10. Blank lines are allowed anywhere in the input file.
11. Lines starting with a `*` are assumed to be comments.
12. Comment lines starting with five or more asterisks (i.e. `*****`) are reproduced verbatim in the output file. These should be used copiously to give a great deal of information about the run being performed, and should be updated before every single run of the code, as it is very easy to lose track of what is being attempted.
13. In-line comments are *usually* ignored, but there can be problems if one contains a comma (`,`). If this is the case, there must also be a comma after the variable's value and before the comment.

It is useful to divide the input file into sections, using suitable comment lines, to help the user keep related variables together.

The following is a valid fragment of an input file (the vertical lines are simply to help show the column alignment):

```
* This line is a comment that will not appear in the output
***** This line is a comment that will appear in the output
boundl(1) = 2.5,
BOUNDU(10) = 3.,
BOUNDU(45) = 1,
* Another comment...  Note that real values can be entered as if
* they were integers, and vice versa (but it's not recommended...)
epsfcn = 10.e-4,
Ftol = 1.D-4,
* The next line sets the first five elements of array icc:
ICC = 2, 10, 11, 24, 31
* The next line sets the first ten elements of array ixc:
ixc = 10, 12, 3, 36, 48,
       1, 2, 6, 13, 16,
IOPTIMZ = 1,
maxcal = 200
 nsweep = 7
NEQNS = 5, This is an in-line comment
NVAR = 10, Another, but successfully containing a comma!
```

The following are *invalid* entries in the input file (Q: Why?!):

```
boundl(1,1) = 2.5,
BOUNDU(N) = 3.,
A line of 'random' characters like this will clearly wreak havoc
eps fcn = 10.e-4, ftol = 1.D-4
epsvmc = 1.0 e-4
ICC = 2 10 11 24 31
IOPTIMZ = 1.0, This will in fact be okay - but is not recommended
NEQNS = 5 An in-line comment on a line with only one comma (,) character
```

If the code encounters a problem reading the input file, it will stop immediately with a (hopefully) useful error message. It may be worth looking at the contents of the output file as well, to help narrow down on which line of the input file the problem might lie.

4.2 The Output File

The main output from the code is sent to file `OUT.DAT` in the working directory.

A second file, `MFILE.DAT`, is also produced in the working directory. This file contains most of the same data as `OUT.DAT` but in a different format, and has been designed to be "machine-readable" by some of the utility programs described in Chapter 6, to allow simple post-processing and graphical output to be produced easily.

4.3 Running the Code

This section will attempt to guide the user through the actual running of the code in its various modes. In most cases only minor changes to the input file are necessary to change the code's mode

of operation — usually the physics and engineering variables, etc. remain unchanged, with the major differences occurring in the numerical input only.

4.3.1 Environment set-up

Under normal circumstances, the code is only available for execution from the CCFE Fusion Unix Network, on which you must have an account.

To set up your environment to be able to run the code and the associated Python utilities (see Chapter 6), add the following lines to your `.bashrc` file:

```
module use /home/pknight/modules
module swap python
module load process/master
```

(If you want to use — with care! — the latest draft version of the code instead of the latest verified release version, replace `module load process/master` with `module load process/develop` above.)

After re-logging in, execute `PROCESS` by simply typing `process` on the command line.

4.3.2 Non-optimisation mode

Non-optimisation mode is used to perform benchmark comparisons, whereby the machine size, output power etc. are known and one only wishes to find the calculated stresses, beta values and fusion powers, for example. When starting to model a new machine, `PROCESS` should always be run first in non-optimisation mode, before any attempt is made to optimise the machine's parameters.

The first thing to do is to add to the input file all the known details about the machine to be modelled. This may include some or all of the following:

- machine build
- plasma aspect ratio
- PF coil locations
- type of current drive to be used
- net electric power
- various physics parameters, e.g.
  - toroidal field on axis
  - electron density
  - electron temperature
  - elongation
  - triangularity
  - beta g coefficient
  - edge safety factor

In addition, some of the switch values summarised in Chapter 3 may have to be altered from their default values.

Next, the relevant numerics information must be entered. Switch `ioptimz` must be set to `-1` for non-optimisation mode. Then the user must decide which constraint equations and iteration variables to activate — this choice is dictated partly by the information required by the user, and partly by the machine being modelled itself.

As stated earlier, all the relevant consistency equations must be activated, together with the corresponding iteration variables. A number of limit equations can also be activated, to investigate how the calculated values compare with the physics or engineering limits. The following is part of an example non-optimisation input file:

```
IOPTIMZ = -1
NEQNS = 8
NVAR = 8
ICC =  1,  2, 10, 11,  7, 16,  5, 24,
IXC =  5, 10, 12, 29,  7,      9, 36, 4,

FPNETEL = 1.0
PNETELIN = 1200.0
:
```

Consistency equations 1, 2, 10, 11 and 7 are activated, together with limit equations 16, 5 and 24 (refer to Table 2.1). This example assumes that neutral beam current drive is present (equation 7 with variable 7), and that the net electric power is to be fixed at 1200 MW. Note the optional (but beneficial) practice of vertically aligning corresponding equations and variables — constraint equation 16 has no corresponding iteration variable (which would normally be no. 25, `fpnetel`), as we want the net electric power to be fixed at the value given by `pnetelin`. Since in non-optimisation mode, the number of variables must be equal to the number of equations, we have scope to add a "free" iteration variable (refer to Tables 2.2 and 2.3), in this case no. 4 — electron temperature, to help raise the fusion power sufficiently to obtain the required net electric power. Finally, note the use of the density and beta limit equations (`5` and `24`, respectively); the final values of the corresponding f-values will indicate if the limits are exceeded and by how much.

On running `PROCESS` and (hopefully) achieving a feasible result, examination of the output may well show up discrepancies between some of the parameter values produced and their known values (if available). Remember that, of all the variables shown in the variable descriptor file with a default value, only those declared as *active iteration variables* can change from their initial values, whether they are set in the input file or in the relevant module. However some of the calculated parameters may be wrong, the most common of which are as follows:

- Plasma current; this can be adjusted using the edge safety factor `q`: $I_p \propto 1/q$
- Fusion power; this scales roughly with the density profile factor `alphan`.
- Build parameters; it may be necessary to change non-critical thicknesses to achieve the correct machine build.

It may still be difficult, if not impossible, to reconcile the fusion power and the net electric power with the required values. This may well be due to the power conversion efficiency values being used — refer to Figure 3.9.

With luck, a few iterations of this process will produce an adequate benchmark case. A typical input file for use with `PROCESS` in non-optimisation mode is contained in Appendix B.

4.3.3 Optimisation mode

Running PROCESS in optimisation mode requires few changes to be made to the input file from the non-optimisation case. The main differences between optimisation mode and non-optimisation mode are:

1. Optimisation mode applies lower and upper bounds to all active iteration variables.
2. There is no upper limit to the number of active iteration variables in optimisation mode.
3. A figure of merit must be specified in optimisation mode.
4. Scans can be performed in optimisation mode.

Switch `ioptimz` must be set to `0` or `1` for optimisation mode. If `ioptimz = 0`, a non-optimisation pass is performed first, to provide a (hopefully) feasible set of initial conditions; if `ioptimz = 1`, this is skipped and the code runs in optimisation mode from the start. However, it is recommended to use `ioptimz = 1` for optimisation runs, as using the HYBRD step before a VMCON iteration tends to cause PROCESS to fail to converge more often.

As before, the user must decide which constraint equations and iteration variables to activate. Again, the choice depends largely on the information required by the user and the extent of the freedom that the code may have with the machine's parameters.

The following is part of an example optimisation input file:

```
IOPTIMZ = 1
NEQNS = 16
NVAR = 19
ICC =  1,  2, 10, 11,  7, 16,  5, 24, 14,  8, 31, 32, 33, 34, 35, 36,
IXC =  5, 10, 12, 29,  7,      9, 36, 19, 14, 48, 49, 50, 51, 53, 54,
4, 6, 1, 18,
BOUNDL(1) = 2.5
BOUNDU(10) = 2.0
MINMAX = 6

FPNETEL = 1.0
PNETELIN = 1200.0
WALALW = 4.4

ISWEEP = 3
NSWEEP = 11
SWEEP = 3.5, 3.7, 3.9
:
```

The figure of merit in this example is the (minimum) cost of electricity (`minmax = 6`). Note that additional limit equations are now active, along with a second consistency equation related to the neutral beam current drive — the number of decay lengths to the plasma centre is constrained to be equal to the input value (`tbeamin`, which is not shown here). Furthermore, there are now more iteration variables than constraint equations, to aid the minimisation process. Finally, note that a three-point scan in the beta g coefficient `dnbeta` — scanning variable 11, is to be performed.

A useful practice in optimisation mode is to perform "stationary" scans, whereby the same value is given to the scanning variable on successive iterations. This provides a check as to how well converged

the solution has become. If scans of a given variable are to be made over a large range of values, it is often a good idea to start the scan in the middle of the desired range, and to split the scan in two — one going downwards from the initial value, and the other upwards. This ensures that the whole range of the scan produces well-converged machines (assuming a "good" initial point), without sharp changes in gradient in the parameter values.

It should be remembered that the value of the scan variable is set in the array `sweep`, and this overrules any value set for the variable elsewhere in the input file. For instance, in the example above, the values of `dnbeta` set in the `sweep` array would overrule any value for `dnbeta` set elsewhere in the file.

The output from an optimisation run contains an indication as to which iteration variables lie at their limiting values. On the whole there is a greater chance of unfeasible solutions being found whilst in optimisation mode, and Section 4.4 will hopefully be of some use in this situation. A typical input file for use with `PROCESS` in optimisation mode is contained in Appendix C.

4.4 Problem Solving

Experience has shown that the first few attempts at running `PROCESS` with a new input file tends to produce unfeasible results — that is, the code will not find a consistent set of machine parameters. The highly non-linear nature of the numerics of `PROCESS` is the reason for this difficulty, and it often requires a great deal of painstaking adjustment of the input file to overcome.

4.4.1 Error handling

In general, errors detected during a run are handled in a consistent manner, with the code producing (hopefully) useful diagnostic messages to help the user understand what has happened.

There are three levels of "error" that may occur:

Level 1: An *informational* message is produced under certain conditions, for example if the code has modified the user's input choice for some reason.

Level 2: A *warning* message is produced if a non-fatal situation has occurred that may result in an output case that is inaccurate or unreliable in some way.

Level 3: An *error* message will occur if a severe or fatal error has occurred and the program cannot continue.

These messages are printed on the screen during the course of a run, and those still active at the final (feasible or unfeasible) solution point are also written to the end of the output file (messages encountered during the iteration process are not copied to the output file, as the convergence to a valid solution might resolve some of the warnings produced earlier in the solution process).

The `error_status` variable returns the highest severity level that has been encountered (or zero if no abnormal conditions have been found); if a severe error (level 3) is flagged at any point the program is terminated immediately. The final message number encountered during a run is returned via output variable `error_id`. In addition, with certain messages, a number of diagnostic values may also be given; these can be used to provide extra diagnostic information if the source code is available.

4.4.2 General problems

A code of the size and complexity of PROCESS contains myriads of equations and variables. Virtually everything depends indirectly on everything else because of the nature of the code structure, so perhaps it is not surprising that it is often difficult to achieve a successful outcome.

Naturally, problems will occur if some of the parameters become unphysical. For example, if the aspect ratio becomes less than or equal to one, then we must expect problems to appear. For this reason, the default bounds on the iteration variables and the allowed ranges of all the input variables have been selected with great care.

The code contains a large (though probably not exhaustive) number of error traps to try and prevent problems from propagating. These include tests for unphysical values, and checks to prevent divisions by zero, and non-sensible arguments for logarithms and square roots, etc. However, occasionally arithmetic ("NaN") errors still occur, although their incidence is low. They now usually only occur due to unfeasibility problems (see later).

The error messages produced by the code attempt to provide diagnostic information, telling the user where the problem occurs, and also suggest a possible solution. These messages are out of necessity brief, and so cannot promise to lead to a more successful outcome.

For expert users, there is the option to turn on extra debugging output; to do this, set `verbose = 1` in the input file.

4.4.3 Optimisation problems

On reflection it is perhaps surprising that PROCESS ever does manage to find the global minimum figure of merit value, since if there are `nvar` iteration variables active the search is over `nvar`-dimensional parameter space, in which there may be many shallow minima of approximately equal depth. Remember that `nvar` is usually of the order of twenty.

The machine found by PROCESS may not, therefore, be the absolutely optimal device. It is quite easy to have two or more solutions, with results only a few per cent different, but a long way apart in parameter space. The technique of "stationary" scans described in Section 4.3.3 above can often help in this situation, which is why this method is recommended at all times.

Scans should be started in the middle of a range of values, to try to keep the scan within the same family of machines. The optimum machine found may otherwise suddenly jump to a new region of parameter space, causing the output variables to seem to vary unpredictably with the scanning variable.

It should be noted that in general the machine produced by PROCESS will always sit against one or more operation limits. If, during a scan, the limit being leant upon changes (i.e. if the machine jumps from leaning on the beta limit to leaning on the density limit) the output parameters may well become discontinuous in gradient, and trends may suddenly change direction.

4.4.4 Unfeasible results

In the numerics section of the output file, the code indicates whether the run produced a feasible or unfeasible result.

The former implies a successful outcome, although it is always worth checking that the estimate of the constraints (`sqsumsq`) is small ($\sim 10^{-3}$ or less); the code will issue a warning if the run seems feasible but the value of `sqsumsq` exceeds 10^{-2}. If this occurs, reducing the value of the HYBRD tolerance

`ftol` or VMCON tolerance `epsvmc` (as appropriate) should indicate whether the result is valid or not; the output can usually be trusted if (1) the constraint residues[1] fall as the tolerance is reduced to about 10^{-8}, and (2) the code indicates that a feasible solution is still found.

An unfeasible result occurs if `PROCESS` cannot find a set of values for the iteration variables which satisfies all the given constraints. In this case, the values of the constraint residues shown in the output give some indication of which constraint equations are not being satisfied — those with the highest residues should be examined further. In optimisation mode, the code also indicates which iteration variables lie at the edge of their allowed range.

Unfeasible runs are caused either by ill-defining the problem to be solved, or by starting the problem in an unfavourable region of parameter space. The latter can be checked simply by changing the initial values of the *active* iteration variables in the input file, but the former requires some extra work. This situation arises if there are insufficient iteration variables for the given constraint equations. It is important to choose the right number of *useful* iteration variables for the problem to be solved — it is possible to activate too many iteration variables as well as too few, some of which may be redundant.

Both optimisation and non-optimisation runs can fail with an error message suggesting that the iteration process is not making good progress. This is likely to be due to the code finding itself unable to escape a region of the parameter space where the minimum in the residuals is significantly above zero. In this situation, there is either no solution possible (the residuals can therefore never approach zero), or the topology of the local minimum makes it difficult for the code to escape to the global minimum. Again, a helpful technique is to either change the list of iteration variables in use, or to simply modify their initial values to try to help the code avoid such regions.

A technique that occasionally removes problems due to unfeasible results, particularly if an error code `ifail = 3` is encountered during an optimisation run, is to adjust slightly one of the limits imposed on the iteration variables, even if the limit in question has not been reached. This subtly alters the gradients computed by the code during the iteration process, and may tip the balance so that the code decides that the device produced is feasible after all. For instance, a certain component's temperature might be 400 K, and its maximum allowable temperature is 1000 K. Adjusting this limit to 900 K (which will make no difference to the *actual* temperature) may be enough to persuade the code that it has found a feasible solution.

Similarly, the order in which the constraint equations and iteration variables are stored in the `icc` and `ixc` arrays can make the difference between a feasible and unfeasible result. This seemingly illogical behaviour is, sadly, typical of the way in which the code works.

Another technique in such situations may be to change the finite difference step length `epsfcn`, as this might subtly change the path taken in the approach towards a solution.

Unfeasible cases often produce unrealistic machines, so one should not believe the output values from these runs. Unfortunately, the stationary scan method sometimes, though not always, fails to help these cases, since it will tend to keep starting the run at the same point. Ill-defined problems sometimes produce arithmetic errors, for obscure reasons.

Though a great deal of work has been performed on the code to improve its standard, there can be no guarantee that `PROCESS` is entirely bug-free, simply because of its large size. Rarely, then, it may be that an unfeasible result indicates that the code has encountered a programming error, although its precise location will be almost impossible to find by simply examining the output file.

It may be the case that the act of satisfying all the required constraints is impossible. No machine can exist if the allowed operating regime is too restrictive, or if two constraint equations require conflicting

[1]The constraint residues are the final values of c_i in the constraint equations — see Section 2.4. The value `sqsumsq` is the square root of the sum of the squares of these residuals.

parameter spaces. In this case some relaxation of the requirements is needed for the code to produce a successful machine design.

4.4.5 Hints

The above sections should indicate that it is the complex interplay between the constraint equations and the iteration variables that determines whether the code will be successful at producing a useful result. It can be a somewhat laborious process to arrive at a working case, and (unfortunately, perhaps) experience is often of great value in this situation.

It should be remembered that sufficient iteration variables should be used to solve each constraint equation. For instance, a particular limit equation may be $A \leq B$, i.e. $A = fB$, where the f-value f must lie between zero and one for the relation to be satisfied. However, if none of the iteration variables have any effect on the values of A and B, and A happens to be *greater* than B, then `PROCESS` will clearly not be able to solve the constraint.

The lower and upper bounds of the iteration variables are all available to be changed in the input file. Constraints can be relaxed in a controlled manner by moving these bounds, although in some cases care should be taken to ensure that unphysical values cannot occur. The code indicates which iteration variables lie at the edge of their range.

It is suggested that constraint equations should be added one at a time, with sufficient new iteration variables activated at each step. If the situation becomes unfeasible it can be helpful to reset the initial iteration variable values to those shown in the output from a previous feasible case, and rerun the code.

Finally, it should be borne in mind that the machine that is envisaged may not be a valid solution to the constraints being imposed, no matter how many degrees of freedom (i.e. iteration variables) are available. In this case, and many others, the user has to relax the constraints slowly until a feasible result is found.

Chapter 5

Inclusion of New Models, Additional Variables and Equations

It is often useful to add extra features to the code in order to model new situations. This chapter provides instructions on how to add various numerics related items to `PROCESS`.

Please remember to modify the relevant Table(s) in this User Guide if changes are made to the source code!

5.1 Input Parameters

Input parameters (see Section 2.3) are added to the code in the following way:

1. Choose the most relevant module (usually one of those in source file `global_variables.f90`). Keeping everything in alphabetical order, add a declaration statement for the new variable, specifying a "sensible" default value, and a correctly formatted comment line to describe the variable (copying the examples already present).

2. Ensure that all the modules that use the new variable reference the relevant module via the Fortran `use` statement.

3. Add the parameter to routine `PARSE_INPUT_FILE` in source file `input.f90` in a suitable place — keep to alphabetical order. The existing examples provide guidance on how to do this. Note that real (i.e. double precision) and integer variables are treated differently, as are scalar quantities and arrays.

4. Modify `process_dicts.py` and `write_constraints.conf` accordingly.

5.2 Iteration Variables

New iteration variables (see Section 2.5) are added in the same way as input parameters, with the following additions:

1. Increment the parameter `ipnvars` in module `numerics` in source file `numerics.f90` to accommodate the new iteration variable.

2. Add an additional line to the initialisation of the array `ixc` in module `numerics` in source file `numerics.f90`.
3. Assign sensible values for the variable's bounds to the relevant elements in arrays `boundl` and `boundu` in module `numerics` in source file `numerics.f90`.
4. Assign the relevant element of character array `lablxc` to the name of the variable, in module `numerics` in source file `numerics.f90`.
5. Add the variable to routines `LOADXC` and `CONVXC` in source file `iteration_variables.f90`, mimicking the way that the existing iteration variables are coded.
6. Modify `process_dicts.py` and `write_constraints.conf` accordingly.

Don't forget to add suitable correctly-formatted comment lines to `numerics.f90` to document the above changes.

If an existing input parameter is now required to be an iteration variable, then simply carry out the tasks mentioned here.

It should be noted that iteration variables must not be reset elsewhere in the code. That is, they may only be assigned new values when originally initialised (in the relevant module, or in the input file if required), and in routine `CONVXC` where the iteration process itself is performed. Otherwise, the numerical procedure cannot adjust the value as it requires, and the program will fail.

5.3 Other Global Variables

This type of variable embraces all those present in the modules in `global_variables.f90` (and some others elsewhere) which do not need to be given initial values or to be input, as they are calculated within the code. These should be added to the code in the following way:

1. Choose the most relevant module (usually one of those in source file `global_variables.f90`). Keeping everything in alphabetical order, add a declaration statement for the new variable, specifying the initial value `0.0D0`, and a correctly formatted comment line to describe the variable (copying the examples already present).
2. Ensure that all the modules that use the new variable reference the relevant module via the Fortran `use` statement.

5.4 Constraint Equations

Constraint equations (see Section 2.4) are added to `PROCESS` in the following way:

1. Increment the parameter `ipeqns` in module `numerics` in source file `numerics.f90` to accommodate the new constraint.
2. Add an additional line to the initialisation of the array `icc` in module `numerics` in source file `numerics.f90`.
3. Assign a description of the new constraint to the relevant element of array `lablcc`, in module `numerics` in source file `numerics.f90`.

4. Add the constraint equation to routine `CONSTRAINTS` in source file `constraint_equations.f90`, ensuring that all the variables used in the formula are contained in the modules specified via `use` statements present at the start of this routine. Use a similar formulation to that used for the existing constraint equations, remembering that the code will try to force `cc(i)` to be zero.

Don't forget to add suitable correctly-formatted comment lines to `numerics.f90` to document the above changes.

Remember that if a limit equation is being added, a new f-value iteration variable may also need to be added to the code.

5.5 Figures of Merit

New figures of merit (see Section 2.6) are added to `PROCESS` in the following way:

1. Increment the parameter `ipnfoms` in module `numerics` in source file `numerics.f90` to accommodate the new figure of merit.
2. Assign a description of the new figure of merit to the relevant element of array `lablmm` in module `numerics` in source file `numerics.f90`.
3. Add the new figure of merit equation to routine `FUNFOM` in source file `evaluators.f90`, following the method used in the existing examples. The value of `fc` should be of order unity, so select a reasonable scaling factor if necessary. Ensure that all the variables used in the new equation are contained in the modules specified via `use` statements present at the start of this file.
4. Modify `process_dicts.py` accordingly.

Don't forget to add suitable correctly-formatted comment lines to `numerics.f90` to document the above changes.

5.6 Scanning Variables

Scanning variables (see Section 2.7) are added to `PROCESS` in the following way:

1. Increment the parameter `ipnscnv` in module `scan_module` in source file `scan.f90` to accommodate the new scanning variable.
2. Add a short description of the new scanning variable to the `nsweep` entry in source file `scan.f90`.
3. Add a new assignment to the relevant part of routine `SCAN` in source file `scan.f90`, following the examples already present, including the inclusion of a short description of the new scanning variable in variable `xlabel`.
4. Ensure that the scanning variable used in the assignment is contained in one of the modules specified via `use` statements present at the start of this routine.
5. Modify `process_dicts.py` if necessary.

5.7 Submission of New Models

The PROCESS source code is maintained by CCFE, and resides in a *Git* [66] repository on the CCFE servers. We welcome contributions of alternative or improved models and algorithms.

The Fortran 90/95 source code has a uniform visual style and structural layout. CCFE will transfer any contributed models into PROCESS, in order for us to maintain its present coding standard. To simplify this task, we request that contributors provide the following information for any new models that they provide:

- A comprehensive description of the model; please provide a full list of references.
- A list of all inputs and outputs: descriptions, default (input) values, allowed ranges, units.
- If possible, please cross-reference any input/output variables to existing global variables listed in the variable descriptor file (see Section 2.2).
- A description of any new numerics requirements (input parameters, iteration variables, constraint equations, figures of merit etc.).
- A definition of any pre-requisites.
- A description of any side-effects.
- Any available test data, code examples or test programs (in Fortran or other language) would be useful.

Chapter 6

Utility Programs

A number of Python utilities for PROCESS are available which allow the user to perform a number of useful actions, for instance to modify the input file IN.DAT, run PROCESS until a feasible solution is found, or to extract and plot data from the PROCESS output. All utilities are available from pknight/process/bin/utilities.

For PROCESS *users*, only the Python executables described in Section 6.1 are relevant. For anyone interested in modifying the existing code or developing new Python utilities the PROCESS Python libraries are described in Section 6.2.

All executables are using Python library functions either from the publicly available numpy, scipy and matplotlib libraries or the PROCESS Python libraries documented in Section 6.2. To use the PROCESS Python libraries you need to make sure their directory is in your Python path. Use the commands given in Section 4.3.1 to do this.

All Python code has been written for Python 3 and adheres to the PEP8 standard.

6.1 Executables

All executables can be run by using any of the following commands:

```
python executable_name.py
python3 executable_name.py
executable_name.py
```

and they typically have a short description of their usage when putting -h or --help as arguments:

```
python executable_name.py -h
```

Some executables come with a configuration file that typically follows the naming convention executable_name.conf. To run the executable please copy the config file into your own work directory. Do **NOT** try to edit the config file in the PROCESS directory.

6.1.1 write_new_in_dat.py

This program modifies a given IN.DAT file such that all iteration variables are given by their values in OUT.DAT. The final values of the iteration variables in OUT.DAT are set in a block at the end of IN.DAT. Any other definitions of these variables are commented out.

Input: IN.DAT, OUT.DAT

Output: new_IN.DAT

6.1.2 write_constraints.py

This Python script modifies the PROCESS input file IN.DAT to include the constraints marked "Y" or "y" in a configuration file called write_constraints.conf. It automatically adds the required iteration variable for each constraint, and any additional iteration variables selected. The script also creates blank upper and lower bound equations; these need to be completed manually or deleted. All files used need to be in the working directory. The original IN.DAT is renamed OLD.IN.DAT.

Input: IN.DAT, write_constraints.conf

Output: IN.DAT, OLD.IN.DAT

Configuration Options: The configuration file write_constraints.conf has the following style:

```
Constraints and associated iteration variables
y ICC=1 plasma beta consistency ixc=5 beta plasma beta 0.001d0 1.0d0
y ICC=2 global power balance    ixc=10 hfact confinement time h-factor 0.1d0 3.0d0
  ICC=3 ion power balance
  ICC=4 electron power balance
...
Remaining iteration variables
  IXC=1 aspect plasma aspect ratio 1.100D0 10.00D0
y IXC=2 bt toroidal field on axis 0.010D0 100.0D0
  IXC=3 rmajor plasma major radius 0.100D0 10.00D0
...
```

An example version that can be modified is available in the utilities directory.

6.1.3 run_process.py

This program runs PROCESS, if necessary, with varying iteration variable start parameters until a feasible solution is found.

Input: run_process.conf, an IN.DAT file as specified in the config file

Output (in the specified work directory): All the standard PROCESS output, process.log (log file), README.txt (contains comments from config file)

Configuration Options: The configuration file run_process.conf has the following style:

```
* This is a comment in the config file!

* Path to working directory in which PROCESS is run.
WDIR = Run1
* original IN.DAT name
ORIGINAL_IN_DAT = demo1a_10_sep_13.IN.DAT
* PATH to PROCESS binary
PROCESS = ~pknight/process/bin/process
* ONE line comment to be put into README.txt
COMMENT = This is a test DEMO run! ;-)
* Max. no iterations
NITER = 5
* integer seed for random number generator; use None for random seed
SEED = 2
* factor within which the iteration variables are changed
FACTOR = 1.5
* Number of allowed unfeasible points that do not trigger rerunning.
NO_ALLOWED_UNFEASIBLE = 0
* include a summary file with the iteration variables at each stage.
INCLUDE_ITERVAR_DIFF = True
* add iteration variables - comma separated
ADD_IXC = 99, 77
* remove iteration variables - comma separated
DEL_IXC =
* add constraint equations  - comma separated
ADD_ICC = 57,
* remove constraint equations - comma separated
DEL_ICC =
* set any variable to a new value
* the variable does not have to exist in IN.DAT
VAR_TFTORT = 1.9
VAR_EPSVMC = 1e-4
* remove variables
DEL_VAR_IITER
```

A configuration file with an alternative name can be specified using the optional argument

```
run_process.py -f CONFIGFILE
```

6.1.4 build_index.py

Creates an index of all `PROCESS` run comments in a series of subfolders.

Input: None

Output: `Index.txt`

Configuration Options: Optional arguments are:

```
# change the name of the file containing the folder description
build_index.py -r README.txt
# append the results to Index.txt instead of creating a new file
build_index.py -m a
# change the name of the subfolder Base - default=Run
build_index.py -b Base
# give a list of subfolder suffixes - default=all
build_index.py -s 1-4,6,8,9-12
# increase verbosity
build_index.py -v
```

An example `Index.txt` file might look like this

```
Run1:
 Original run

Run2:
 Changed the no. TF coils
...
```

6.1.5 make_plot_dat.py

Creates a `PLOT.DAT`-type file from `MFILE.DAT`.

Input: `make_plot_dat.conf`, `MFILE.DAT`

Output: `make_plot_dat.out`

Configuration Options: Optional arguments are:

```
# new variables for output
make_plot_dat.py -p rmajor
# writes make_plot_dat.out in columns
make_plot_dat.py --columns
# resets make_plot_dat.conf to PLOT.DAT layout
make_plot_dat.py --reset-config
# file to read as input
make_plot_dat.py -f MFILE.DAT
# run with default parameters
make_plot_dat.py --defaults
```

An example version of `make_plot_dat.conf` might look like this:

```
# make_plot_dat.out config file.
rmajor
aspect
rminor
bt
powfmw
pnetelmw
te
pdivt
strtf1
strtf2
```

6.1.6 plot_proc.py

A utility to produce the standard PROCESS summary output, covering the major parameters, provenance data, and a machine cross-section. It allows a wide range of graphical output formats and can easily be customised using the function `gather_info()` in `plot_proc_func.py`.

Input: `MFILE.DAT`

Output: `proc_plot_out.pdf` (or as specified by user)

Configuration Options: Optional arguments are:

```
# change the input file name
python plot_proc.py -f MFILE.DAT
# change the output file name
python plot_proc.py -o out.pdf
# show the plot as well as saving the figure
python plot_proc.py -s
```

6.1.7 plot_sweep.py

This utility plots normalised values from `PLOT.DAT` and allows comparisons of changes in values from a sweep compared to the first point in the sweep. It is now somewhat deprecated due to the existence of more flexible sweep-generating and output-handling utilities.

Input: `PLOT.DAT`

Output: `PLOT.DAT.eps` (default or as specified by user)

Configuration Options: Optional arguments are:

```
# creates PLOT.DAT.eps with R0, IP, beta
python plot_sweep.py -g 11 13 18
```

```
# creates demo1.png with Te, n
python plot_sweep.py -o demo1.png 21 22
```

6.1.8 plot_mfile_sweep.py

This utility plots normalised values from `MFILE.DAT` and allows comparisons of changes in values from a sweep compared to the first point in the sweep.

Input: `MFILE.DAT`

Output: `sweep_fig.pdf` (default or as specified by user)

Optional arguments are:

```
# creates sweep_fig.pdf with R0, te, aspect (same variable names as in MFILE.DAT)
python plot_mfile_sweep.py -p rmajor te aspect
# creates demo1.png with Te, n
python plot_mfile_sweep.py -o demo1.png -p te dene
# creates a sweep_fig.pdf with R0, aspect with a different MFILE.DAT
python plot_mfile_sweep.py -f diff_mfile.dat -p rmajor aspect
# Show plot to screen instead of saving with R0 and aspect
python plot_mfile_sweep.py -p rmajor aspect --show

Use -h or --help for help
```

6.1.9 diagnose_process.py

This utility aids the user to interpret (unsuccessful) `PROCESS` runs, unless `PROCESS` has terminated prematurely. It takes an existing `MFILE.DAT` and plots the normalised iteration variables, i.e. the iteration variable values normalised to their bounds such that 0 indicates an iteration variable at its lower bound and 1 an iteration variable at its upper bound. Furthermore, it shows the normalised constraint residuals.

Input: `MFILE.DAT`

Output: Displays plots on screen, still need to be saved by the user! (Remember to use -Y or -X, if `ssh`ing into a remote machine!)

Optional arguments are:

```
# allows to specify another location/name for the MFILE
python diagnose_process.py -f MFILE.DAT

Use -h or --help for help
```

6.1.10 2D_scan.py

N.B. This utility is currently not working — please do not try to use it.

Runs a two-dimensional scan of PROCESS variables. This code does not yet use the new PROCESS Python libraries.

Input: 2D_scan.conf, IN.DAT

Output: All the standard PROCESS output files, ERROR.DAT (Error matrix for all iterations), EDGETABLE.DAT (Marks if variables at bounds), MATRICES.DAT (Solution for each variable), EDGEVARS.DAT (Names of variables at bounds)

Configuration Options: The configuration file 2D_scan.conf has the following style

```
* This is a comment

* No evaluations of the first variable
N1= 10
* lower bound of first variable
LB1= 1
* upper bound of first variable
UB1= 3
* name of first variable
NAME1= abktflnc

* No evaluations of the second variable
N2= 15
* lower bound of the second variable
LB2= 3
* upper bound of the second variable
UB2= 8
* name of the second variable
NAME2= bktlife

* Output variables as given by their variable description
OUT_VARS
Major Radius
Pdivt
/OUT_VARS

* Directory path to IN.DAT
PATH=/DIR2INDAT
* Boolean flag to store OUT.DAT and PLOT.DAT for each run
* in the Data/ subdir
DATA=0
```

6.1.11 a_to_b.py

Takes an initial IN.DAT and a target IN.DAT and runs PROCESS repeatedly with the aim of walking the solution to the initial input file to a solution using the values of the variables in the target input file. After each step, the output values of the iteration variables are used as the input values for the next step.

Input: a_to_b.conf

Output

When the program finishes, the .DAT files from the last run of PROCESS can be found in the specified working directory.

If option keep_output = True: IN.DAT, OUT.DAT, MFILE.DAT and logged process output of each step are stored. Files are prefaced with their step number eg. 003.IN.DAT and copied to the specified output directory.

Configuration Options: The configuration file a_to_b.conf has the following style:

```
*Comment line

*Working directory to store temporary files, default = wdir
wdir = wdir

*Switch to keep .DAT files for every step, default = True
keep_output = True

*Directory for output if keep_output = True, default = steps
outdir = steps

*IN.DAT file for A, default = A.DAT
a_filename = A.DAT

*IN.DAT file for B, default = B.DAT
b_filename = B.DAT

*Path to process binary
*path_to_process = /home/pknight/process/bin/process
path_to_process = /home/pknight/process/bin/process

*Number of iterations of vary_iteration_variables to run, default = 20
vary_niter = 20

*Number of steps to go from A to B, default = 10
nsteps = 10

*Factor to vary iteration variables within, default = 1.2
factor = 1.2
```

```
*Gap between upper and lower bounds to narrow to, default = 1.001
bound_gap = 1.001
```

6.1.12 create_dicts.py

To do

6.2 Libraries

All library modules and functions are documented using docstrings. These can be accessed by reading the code directly or via the `help()` function in Python.

6.2.1 in_dat.py

A set of Python classes to read, modify and write an `IN.DAT` file.

`INVariable(name, value, comment="")` Initialises an `IN.DAT` variable class which requires a name and a value.

`INModule(name)` Initialises an `IN.DAT` module class which requires a name. This class stores information for an `IN.DAT` file which has modules separated with lines with $MODULE_NAME and $END.

`INModule.add_variable(var)` Adds an `INVariable` object to the `INModule` class.

`INModule.remove_variable(variable_name)` Removes an `INVariable` object from the `INModule` class.

`INModule.add_line(line)` Adds a line from the `IN.DAT` that isn't a variable line (e.g. a comment) to the `INModule` class.

`INModule.get_var(var_name)` Returns an `INVariable` object from the `INModule` class.

`INModule.add_constraint_eqn(eqn_number)` Adds constraint equation `eqn_number` to the list of constraint equations `InVariable` class if the equation is not already in the list.

`INModule.remove_constraint_eqn(eqn_number)` Removes constraint equation `eqn_number` from the list of constraint equations `InVariable` class if the equation is already in the list.

`INModule.add_iteration_variable(var_number)` Adds iteration variable `var_number` to the list of iteration variables `InVariable` class if the variable is not already in the list.

`INModule.remove_iteration_variable(var_number)` Removes iteration variable `var_number` to the list of iteration variables `InVariable` class if the variable is already in the list.

`INDATClassic(filename="IN.DAT")` Initialises an `IN.DAT` class which can be given a different filename. This class is used for an `IN.DAT` with a module structure.

`INDATClassic.read_in_dat()` Reads an `IN.DAT` file with modular structure.

`INDATClassic.write_in_dat(filename="new_IN.DAT")` Writes a new `IN.DAT` with modular structure.

`INDATNew.read_in_dat()` Reads an `IN.DAT` file without modular structure.

`INDATNew.write_in_dat(filename="new_IN.DAT")` Writes a new `IN.DAT` without modular structure.

`INDATNew.add_variable(var)` Adds an `INVariable` object to the `INDATNew` class.

`INDATNew.remove_variable(var_name)` Removes an `INVariable` object from the `INDATNew` class.

`INDATNew.add_constraint_eqn(eqn_number)` Adds constraint equation `eqn_number` to the list of constraint equations `InVariable` class if the equation is not already in the list.

`INDATNew.remove_constraint_eqn(eqn_number)` Removes constraint equation `eqn_number` from the list of constraint equations `InVariable` class if the equation is already in the list.

`INDATNew.add_iteration_variable(var_number)` Adds iteration variable `var_number` to the list of iteration variables `InVariable` class if the variable is not already in the list.

`INDATNew.remove_iteration_variable(var_number)` Removes iteration variable `var_number` to the list of iteration variables `InVariable` class if the variable is already in the list.

`clear_lines(lines)` Removes comment only lines and replaces multiple empty lines with a single empty line.

`variable_type(var_name, var_value)` Checks the type of an `IN.DAT` variable using `DICT_VAR_TYPE`

`fortran_python_scientific(var_value)` Changes from FORTRAN double precision notation `D` to Python's `e` notation.

6.2.2 mfile.py

A set of Python classes to read and extract data from `MFILE.DAT`.

`MFileVariable(var_name, var_description)` Object to contain information for a single `MFILE.DAT` variable.

`MFileVariable.set_scan(scan_number, scan_value)` Sets the variable value for scan number `scan_number`

`MFileVariable.get_scan(scan_number)` Returns the variable value for scan number `scan_number`

`MFileVariable.get_scans()` Returns the variable value for all scans.

`MFile(filename="MFILE.DAT")` Object to contain information for all variables in an `MFILE.DAT` for all scans.

`MFile.search_keys(variable)` Search for an MFILE variable in the data dictionary.

`MFile.search_des(description)` Search for an MFILE description in the data dictionary.

`MFile.make_plot_dat(custom_keys, filename="make_plot_dat.out", file_format="row")` Create a `PLOT.DAT` equivalent for the variables in `custom_keys` in either row or column format.

`MFile.get_num_scans()` Returns the number of scans in the `MFILE.DAT`

`read_mplot_conf(filename="make_plot_dat.conf")` Reads the config file `make_plot_dat.conf` and fills the list `custom_keys` with these variables.

`write_mplot_conf(filename="make_plot_dat.conf")` Writes a new `make_plot_dat.conf` adding any additional variables that the user `gave make_plot_dat.py` at runtime.

6.2.3 process_funcs.py

This library contains a collection of functions used by various `PROCESS` utilities.

`get_neqns_itervars(wdir='.')` Returns the number of equations and a list of variable names of all iteration variables.

`update_ixc_bounds(wdir='.')` Updates the lower and upper bounds in `DICT_IXC_BOUNDS` from `IN.DAT`.

`get_variable_range(itervars, factor, wdir='.')` Returns the lower and upper bounds of the variable range for each iteration variable.

`itervars`: string list of all iteration variable names

`factor`: defines the variation range for non-f-values by setting them to value * factor and value / factor respectively while taking their `PROCESS` bounds into account.

For f-values the allowed range is equal to their `PROCESS` bounds.

`check_logfile(logfile='process.log')` Checks the log file of the `PROCESS` output. Stops if an error occurred that needs to be fixed before rerunning.

`process_stopped(logfile='process.log')` Checks the `PROCESS` logfile whether it has prematurely stopped.

`process_warnings(logfile='process.log')` Checks the `PROCESS` logfile whether any warnings have occurred.

`mfile_exists()` Checks whether `MFILE.DAT` exists.

`no_unfeasible_mfile(wdir='.')` Returns the number of unfeasible points in a scan in `MFILE.DAT`.

`no_unfeasible_outdat(wdir='.')` Returns the number of unfeasible points in a scan in `OUT.DAT`.

`vary_iteration_variables(itervars, lbs, ubs)` Changes the iteration variables in `IN.DAT` within given bounds.

`itervars`: string list of all iteration variable names

`lbs`: float list of lower bounds for variables

`ubs`: float list of upper bounds for variables

`get_solution_from_mfile(neqns, nvars, wdir='.')` Returns:-

- `ifail`: error value of `PROCESS`
- the objective functions
- the square root of the sum of the squares of the constraints
- a list of the final iteration variable values
- a list of the final constraint residue values
- If the run was a scan, the values of the last scan point will be returned.

`get_solution_from_outdat(neqns, nvars)` Returns:-

- `ifail`: error value of `PROCESS`
- the objective functions
- the square root of the sum of the squares of the constraints
- a list of the final iteration variable values
- a list of the final constraint residue values
- If the run was a scan, the values of the last scan point will be returned.

6.2.4 process_config.py

A collection of Python classes for configuration files used by various `PROCESS` utilities, e.g. `run_process.py` or `test_process.py`. It contains a base class `ProcessConfig` and two derived classes `RunProcessConfig` and `TestProcessConfig`.

`ProcessConfig()` Object that contains the configuration parameters for PROCESS runs.

`filename` : Configuration file name

`wdir` : Working directory

`or_in_dat` : Original `IN.DAT` file

`process` : `PROCESS` binary

`niter` : (Maximum) number of iterations

`u_seed` : User specified seed value for the random number generator

`factor` : Multiplication factor adjusting the range in which the original iteration variables should be varied

`comment` : Additional comment to be written into `README.txt`

`ProcessConfig.echo_base()` Echos the attributes of the base class to standard output.

`ProcessConfig.echo()` Echos the values of the current class to standard output.

`ProcessConfig.prepare_wdir()` Prepares the work directory for the run.

`ProcessConfig.create_readme(directory='.')` Creates a file called `README.txt` containing `ProcessConfig.comment`.

`ProcessConfig.modify_in_dat()` Modifies the original `IN.DAT` file.

`ProcessConfig.setup()` Sets up the program for running.

`ProcessConfig.run_process()` Runs `PROCESS` binary.

`ProcessConfig.get_comment()` Gets the comment line from the configuration file.

`ProcessConfig.get_attribute(attributename)` Gets a class attribute from the configuration file.

`ProcessConfig.set_base_attributes()` Sets the attributes of the base class.

`TestProcessConfig(filename='test_process.conf')` Object that contains the configuration parameter of the `test_process.py` program.

`ioptimz` : sets `ioptimz` (optimisation solver) in `IN.DAT`.

`epsvmc` : sets `epsvmc` (VMCON error tolerance) in `IN.DAT`.

`epsfcn` : sets `epsfcn` (finite diff. steplength) in `IN.DAT`.

`minmax` : sets `minmax` (figure of merit switch) in `IN.DAT`.

`TestProcessConfig.echo()` Echos the values of the current class to std out.

`TestProcessConfig.modify_in_dat()` Modifies `IN.DAT` using the configuration parameters.

`RunProcessConfig(filename='run_process.conf')` Configuration parameters of the `run_process.py` program.

`no_allowed_unfeasible` : The number of allowed unfeasible points in a sweep

`create_itervar_diff` : Boolean to indicate the creation of a summary file of the iteration variable values at each stage

`add_ixc` : List of iteration variables to be added to `IN.DAT`.

`del_ixc` : List of iteration variables to be deleted from `IN.DAT`.

`add_icc` : List of constrained equations to be added to `IN.DAT`.

`del_icc` : List of constrained equations to be deleted from `IN.DAT`.

`dictvar` : Dictionary mapping variable name to new value (replaces old or gets appended)

`del_var` : List of variables to be deleted from `IN.DAT`.

`RunProcessConfig.get_attribute_csv_list(attributename)` Get a class attribute list from the configuration file; expects comma separated values.

`RunProcessConfig.set_del_var()` Sets the `RunProcessConfig.del_var` attribute from the config file.

`RunProcessConfig.set_dictvar()` Sets the `RunProcessConfig.dictvar` attribute from config file.

`RunProcessConfig.echo()` Echos the values of the current class.

`RunProcessConfig.modify_in_dat()` Modifies `IN.DAT` using the configuration parameters.

`RunProcessConfig.modify_vars()` Modifies `IN.DAT` by adding, deleting and modifiying variables.

`RunProcessConfig.modify_ixc()` Modifies the array of iteration variables in `IN.DAT`.

`RunProcessConfig.modify_icc()` Modifies the array of constraint equations in `IN.DAT`.

6.2.5 process_dicts.py

A collection of dictionaries and lists used by various `PROCESS` utilities.

`IFAIL_SUCCESS` This is the `PROCESS` error code of a successful run.

`PARAMETER_DEFAULTS` Default values for making a `PLOT.DAT` file from `MFILE.DAT`.

`DICT_VAR_TYPE` Maps the `PROCESS` variable name to its value type. The value type can be one of `int_array`, `int_variable`, `real_array` or `real_variable`.

`DICT_IXC_SIMPLE` Maps the string number of the iteration variable to its variable name.

`DICT_IXC_FULL` Maps the string number of the iteration variable to a dictionary that contains the variable name under 'name', the default lower variable bound (float) under 'lb' and the default upper variable bound (float) under 'ub'.

`DICT_IXC_BOUNDS` Maps each iteration variable name to a dictionary that contains the default lower variable bound (float) under 'lb' and the default upper variable bound (float) under 'ub'.

`NON_F_VALUES` List of iteration variable names that start with an f, but are not f-values.

`DICT_NSWEEP2IXC` Maps the sweep variable number `nsweep` to the respective iteration variable number, if applicable.

`DICT_IX2NSWEEPC` Maps the iteration variable number to the respective sweep variable number `nsweep`, if applicable.

`DICT_TF_TYPE` Maps the `PROCESS` TF coil type number to its type.

`DICT_OPTIMISATION_VARS` Maps each figure of merit number to its description.

`DICT_IXC_DEFAULT` Maps each iteration variable name to its default value.

Add new stuff used by GUI etc.

6.2.6 a_to_b_config.py

To do...

6.2.7 proc_plot_func.py

A collection of functions and lists used by `plot_proc.py`.

`RADIAL_BUILD` A list of radial build variables.

`VERTICAL_BUILD` A list of vertical build variables.

`FILL_COLS` A list of plotting colours

For all of the functions below the arguments are:

`axis` : Matplotlib axis object to plot to

`mfile_data` : MFILE data object `MFile`

`scan` : Scan number to plot

`plot_plasma(axis, mfile_data,scan)` Function to plot plasma

`plot_machine_pic(axis, mfile_data, scan)` Function to plot machine build

`plot_tf_coils(axis, mfile_data, scan)` Function to plot the TF coils

`plot_pf_coils(axis, mfile_data, scan)` Function to plot the PF coils

`plot_geometry_info(axis, mfile_data, scan)` Function to plot the geometry info block

`plot_physics_info(axis, mfile_data, scan)` Function to plot the physics info block

`plot_magnetics_info(axis, mfile_data, scan)` Function to plot the magnetics info block

`plot_power_info(axis, mfile_data, scan)` Function to plot the power info block

`plot_current_drive_info(axis, mfile_data, scan)` Function to plot the current drive info block

`plot_geometry_info(axis, mfile_data, scan)` Function to plot the geometry info block

6.3 User Interface

The user interface is still being developed, but currently allows for viewing and editing of `IN.DAT` files in a web browser. It can be found in the `utilities/gui` directory.

6.3.1 Launching

The interface uses Django for its web framework. Currently it can only be launched by running the Django development server locally. This is performed automatically if you launch the GUI by typing `start_gui` from the command line. For instructions on starting the server manually, see the `README` file in the `utilities/gui` directory.

6.3.2 Editing variables

Every variable should be listed along with its current value, a 'reference' value and a short description. Hovering over the short description will provide a longer description. All values begin with the default value assigned by `PROCESS`.

The reference values are used to compare differences between two files. Differences between the reference value and the current input value will be highlighted in red.

The 'Meta' section allows editing of the run description, which will be placed at the top of the created `IN.DAT`. Changes to the iteration variables and constraint equations that are enabled can be done using the checkboxes in the relevant section.

6.3.3 Opening and saving files

An input file and a reference file can be opened using the buttons at the top of the screen. The files opened must be compatible with the `PROCESS` version number shown in the top right of the screen.

The 'Save' button should produce a `IN.DAT` file with a similar layout to the GUI. Variables whose values are set different from the `PROCESS` default are listed under module headings, along with a comment describing the variable. For integer variables, a description of the value taken may also be included. The variables `neqns` and `nvar` are automatically calculated.

6.3.4 Searching

There is currently no search feature, but searching for a particular variable can be done using the browser's in-built search. Use the button in the top-right of the screen to expand every module heading and use Ctrl-f to search through the page.

Chapter 7

Code Management Tools

This chapter will be of interest to people involved in the continuing maintenance of the `PROCESS` source code. As stated elsewhere, the source code is maintained by CCFE, and resides in a *Git* repository on the CCFE servers.

7.1 The Makefile

In addition to its normal role for compilation, the makefile (named `Makefile`) includes a number of utility functions that perform tasks such as automatic generation of the code documentation, and the creation of a tar file containing the entire source code, its documentation files, and the input and output files. This has proved to be of great benefit in keeping all of the data from a given run together for archival purposes.

7.1.1 Compilation

Compilation is trivially performed using the makefile. Currently, the available options are (type the following on the command line):

`make ARCH=FUN`	CCFE Fusion Unix Network (Intel Fortran ifort compiler)
`make ARCH=GFORT`	GNU Fortran compiler (gfortran; N.B. only versions 4.6.3 or above)
`make ARCH=JAC`	JET Analysis Cluster (pgf95 compiler)

The Intel Fortran compiler is the default option, so typing simply `make` will have the same effect as typing `make ARCH=FUN`.

The code is almost trivial to port to new architectures, by adding extra stanzas to the makefile as required.

To force a full recompilation, type `make clean`. N.B. This will remove (without prompting) all "backup" files (`*~`) as well as certain other files from the working directory.

Extra run-time diagnostics (array-bound checking, etc.) is turned on by default at present; the code is sufficiently quick to run that the performance penalty is barely noticeable. However, this can be turned off (beware...results can change!), by adding `DEBUG=NO` to the relevant command above, i.e. type `make ARCH=... DEBUG=NO`. N.B. there must be no spaces either side of the `=` characters in any of the above commands.

7.1.2 Archiving utilities

The makefile can be used in a number of ways to pack the various file sets together into a single compressed tar file, for ease of portability and archiving.

- `make tar`: Typing this command produces a file `process.tar.gz` that contains all of the source files, utility programs and documentation files necessary to run the program from scratch on a new machine or in a new directory. (Note that the input files `IN.DAT` and `device.dat` are not included.) This is useful for transferring new copies of source files, etc. into an existing directory already containing a previous `PROCESS` run, as the pre-existing customised input files will not be overwritten. To extract the individual files again, copy the file to the destination working directory and type `tar zxvf process.tar.gz`
- `make archive`: Typing this command produces a file `process_run.tar.gz` containing the working directory's input and output files (`IN.DAT`, `OUT.DAT`, `PLOT.DAT`, `MFILE.DAT` and `device.dat` (which must exist to avoid an error message; see Section 4.1.1), together with all of the source files, utility programs and documentation files. These files together comprise the full set that define a given run, and so the file produced by this command is suitable for long-term storage for archival purposes. To extract the contents, type `tar zxvf process_run.tar.gz`

Note that each of the above commands will overwrite the given named tar file if one already exists in the working directory.

7.1.3 Code documentation

The makefile can also be used to produce source code documentation, as described in the next section.

7.2 Automatic Documentation

The `PROCESS` source code is self-documenting to a degree, using an included parser program (`autodoc`) to generate html files for each subprogram from specially-formatted comment lines within the code. It is the responsibility of the programmer to keep the `autodoc` comments within the source code relevant, comprehensive and up-to-date! Use the examples in the code as a starting basis for new routines; the output section corresponding to the various `autodoc` tags should be self-explanatory. The following files are used:

```
autodoc.f90
adheader.src
adfooter.src
```

To create the ($\sim$ 380) html files from the source code, type `make html`. Then, point your favourite web browser to the file `process.html` or `calling_tree.html` in the working directory. The variable descriptor file `vardes.html` (see Section 2.2) is also produced at the same time (though it is best to modify the file manually afterwards to remove empty sections from it).

In addition, a full LaTeX User Guide `process.tex` (this document!) is rigorously maintained to ensure its continuing strict agreement with the evolving source code. To create an Adobe PDF file `process.pdf` from `process.tex` (and the associated PostScript figures), type `make manual`.

Finally, to produce both the html files and the User Guide in one go, simply type `make doc`.

Chapter 8

Acknowledgements & Bibliography

The author would like to thank the following people for many useful and revealing discussions during his work on PROCESS:

— John D. Galambos, Paul C. Shipe and Y-K. Martin Peng of Oak Ridge National Laboratory, USA;

— Ian Cook, Robin Forrest, Winston Han, Roger Hancox and Panos Karditsas, all formerly of Fusion Physics Department, UKAEA Fusion;

— John Hicks, formerly of Engineering Department, UKAEA Fusion;

— Chris Gardner, formerly of Microwave and Interpretation Department, UKAEA Fusion;

— Tim Hender of CCFE, and all the co-authors of reference [15];

— David Ward, Richard Kemp, Michael Kovari, Hanni Lux and James Morris, all of CCFE, and the members of the PROCESS User Group throughout Europe.

This work was funded by the RCUK Energy Programme under grant EP/I501045 and the European Communities under the contract of Association between EURATOM and CCFE. The views and opinions expressed herein do not necessarily reflect those of the European Commission. Part of this work was carried out within the framework of the European Fusion Development Agreement.

Bibliography

[1] R. L. Reid et al., *"ETR/ITER Systems Code"*, Oak Ridge Report ORNL/FEDC-87/7 (1988)

[2] J. D. Galambos, *"STAR Code : Spherical Tokamak Analysis and Reactor Code"*, Unpublished internal Oak Ridge document. A copy exists in the PROCESS Project Work File [31].

[3] J. J. More, B. S. Garbow and E. Hillstrom, *"User Guide for MINPAC-1"*, Argonne National Laboratory Report ANL-80-74 (1980)

[4] M. J. D. Powell, *"A Hybrid Method for Non-linear Algebraic Equations"*, Numerical Methods for Non-linear Algebraic Equations, ed. P. Rabinowitz, Prentice-Hall

[5] R. L. Crane, K. E. Hillstrom and M. Minkoff, *"Solution of the General Nonlinear Programming Problem with Subroutine VMCON"*, Argonne National Laboratory Report ANL-80-64 (1980)

[6] M. J. D. Powell, *"A Fast Algorithm for Nonlinearly Constrained Optimization Calculations"*, Lecture Notes in Mathematics, vol. 630, pp.144–157, Springer-Verlag, Berlin, 1978

[7] M. Avriel, *"Nonlinear Programming: Analysis and Methods"*, Dover Publications, Inc., Mineola, NY, 2003

[8] P. J. Knight, *"Surface Area and Volume Calculations for Toroidal Shells"*, CCFE internal note T&M/PKNIGHT/PROCESS/009, May 2013

[9] H. Zohm et al, *"On the Physics Guidelines for a Tokamak DEMO"*, FTP/3-3, Proc. IAEA Fusion Energy Conference, October 2012, San Diego

[10] T. Hartmann and H. Zohm, *"Towards a 'Physics Design Guidelines for a DEMO Tokamak' Document"*, EFDA Report, March 2012 (Activity_3_Physics_Design_Guidelines_2L8QVN_v1_0(1).pdf)

[11] P. J. Knight, CCFE Logbook, F/MI/PJK/LOGBOOK14, pp.41–43

[12] H.-S. Bosch and G. M. Hale, *"Improved Formulas for Fusion Cross-sections and Thermal Reactivities"*, Nuclear Fusion **32** (1992) 611

[13] J. Johner, *"Helios: A Zero-Dimensional Tool for Next Step and Reactor Studies"*, Fusion Science and Technology **59** (2011) 308–349

[14] N. A. Uckan and ITER Physics Group, *"ITER Physics Design Guidelines: 1989"*, ITER Documentation Series, No. 10, IAEA/ITER/DS/10 (1990)

[15] T. C. Hender, M. K. Bevir, M. Cox, R. J. Hastie, P. J. Knight, C. N. Lashmore-Davies, B. Lloyd, G. P. Maddison, A. W. Morris, M. R. O'Brien, M. F. Turner and H. R. Wilson, *"Physics Assessment for the European Reactor Study"*, AEA Fusion Report AEA FUS 172 (1992)

[16] M. Kovari et al., *"PROCESS: a systems code for fusion power plants - Part 1: Physics"*, in preparation (2014)

[17] H. Lux et al., *"Impurity radiation in DEMO"*, in preparation (2014)

[18] Albajar, Nuclear Fusion **41** (2001) 665

[19] Fidone, Giruzzi and Granata, Nuclear Fusion **41** (2001) 1755

[20] N. A. Uckan, Fusion Technology **14** (1988) 299

[21] W. M. Nevins et al., *"Summary Report: ITER Specialists' Meeting on Heating and Current Drive"*, ITER-TN-PH-8-4, 13–17 June 1988, Garching, FRG

[22] H. R. Wilson, Nuclear Fusion **32** (1992) 257

[23] O. Sauter, C. Angioni and Y. R. Lin-Liu, Physics of Plasmas **6** (1999) 2834

[24] O. Sauter, C. Angioni and Y. R. Lin-Liu, Physics of Plasmas **9** (2002) 5140

[25] N. A. Uckan et al., Fusion Technology **13** (1988) 411

[26] A. Li Puma, F. Franza and L. V. Boccaccini, *"WP12-SYS01-T02 - Model Improvements (Blanket Model)"*, EFDA_D_2LKMCT, v1.0, EFDA Power Plant Physics & Technology, February 2013

[27] P. J. Karditsas, *"`PROCESS` Code Development: simple blanket model — neutron heat input"*, Work File Note F/RS/CIRE5523/PWF/0142, November 1992

[28] L. Bottura, *"$J_c(B, T, \epsilon)$ Parameterizations for the ITER Nb_3Sn Production"*, ITER Document 2MMF7J (2008), `https://user.iter.org/?uid=2MMF7J&action=get_document`

[29] J. Myall, ORNL internal note, 28th August 1987; scanned copy available in `Myall_1987_TF_stress_calc.pdf`

[30] J. Morris, *"PROCESS Superconducting Toroidal Field Coil Model"*, CCFE internal note, 1st May 2014

[31] P. J. Knight, *"`PROCESS` Reactor Systems Code"*, AEA Fusion Project Work File, F/RS/CIRE5523/PWF (1992)

[32] Y-K. M. Peng and J. B. Hicks, *"Engineering Feasibility of Tight Aspect Ratio Tokamak (Spherical Torus) Reactors"*, AEA Fusion Report AEA FUS 64 (1990)

[33] *"Pulsed Fusion Reactor Study"*, AEA Fusion Report AEA FUS 205 (1992)

[34] F. Schauer, K. Egorov and V. Bykov, *"HELIAS 5-B magnet system structure and maintenance concept"*, Fusion Engineering and Design `88` (2013) 1619–1622

[35] F. Warmer, *"Stellarator Plasma Geometry model for the systems code PROCESS"*, IPP Greifswald, Germany, internal note, 19/06/2013

[36] F. Warmer, *"Stellarator Divertor model for the systems code PROCESS"*, IPP Greifswald, Germany, internal note, 21/06/2013

[37] F. Warmer and F. Schauer, *"Stellarator Coil model for the systems code PROCESS"*, IPP Greifswald, Germany, internal note, 07/10/2013

[38] VMEC MHD force balance code for toroidal domains, `http://vmecwiki.pppl.wikispaces.net/VMEC`

[39] J. Geiger, *"Darstellung von ineinandergeschachtelten toroidal geschlossenen Flächen mit Fourierkoeffizienten" "Representation of nested, closed surfaces with Fourier coefficients"* IPP Greifswald, Germany, internal document, 06/07/2010

[40] J. Nührenberg et al., *Plasma Physics and Controlled Fusion*, **35** (1993) B115

[41] S. Sudo, Y. Takeiri, H. Zushi et al., *Nuclear Fusion*, **30** (1990) 11

[42] R. J. Goldston, H. Biglari, G. W. Hammett et al., *Bull. Am. Phys. Society*, **34** (1989) 1964

[43] K. Lackner and N. A. O. Gottardi, *Nuclear Fusion*, **30** (1990) 767

[44] U. Stroth et al., *Nuclear Fusion*, **36** (1996) 1063

[45] H. Yamada et al., *Nuclear Fusion*, **45** (2005) 1684

[46] F. Warmer, *"Stellarator Modular Coil model for the systems code PROCESS"*, IPP Greifswald, Germany, internal note, 31/07/2013

[47] *"The TITAN Reversed Field Pinch Fusion Reactor Study, Scoping Phase Report"* , UCLA Report UCLA-PPG-1100, January 1987

[48] *"The TITAN Reversed Field Pinch Fusion Reactor Study, Final Report"*, UCLA Report UCLA-PPG-1200, 1990

[49] P. J. Knight, *"PROCESS 3009: Incorporation of Inertial Fusion Energy Model"*, Work File Note F/MI/PJK/PROCESS/CODE/032

[50] Bourque et al., *"Overview of the OSIRIS IFE Reactor Conceptual Design"*, Fusion Technology **21** (1992) 1465

[51] Meier and Bieri, *"Economic Modeling and Parametric Studies for OSIRIS — a HIB-Driven IFE Power Plant"*, Fusion Technology **21** (1992) 1547

[52] Ghose et al., *"BOP Designs for OSIRIS and SOMBRERO IFE Reactor Plants"*, Fusion Technology **21** (1992) 1501

[53] Sviatoslavsky et al., *"A KrF Laser Driven Inertial Fusion Reactor SOMBRERO"*, Fusion Technology **21** (1992) 1470

[54] Meier and Bieri, *"Economic Modeling and Parametric Studies for SOMBRERO — a Laser-Driven IFE Power Plant"*, Fusion Technology **21** (1992) 1552

[55] Moir et al., *"HYLIFE-II: A Molten-Salt Inertial Fusion Energy Power Plant Design — Final Report"*, Fusion Technology **25** (1994) 5

[56] Moir, *"HYLIFE-II Inertial Fusion Energy Power Plant Design"*, Fusion Technology **21** (1992) 1475

[57] Hoffman and Lee, *"Performance and Cost of the HYLIFE-II Balance of Plant"*, Fusion Technology **21** (1992) 1475

[58] P. J. Knight, *"PROCESS IFE Build Details"*, F/MI/PJK/LOGBOOK12, pp. 52, 53, 56, 57

[59] Bieri and Meier, *"Heavy-Ion Driver Parametric Studies and Choice of a Base 5 MJ Driver Design"*, Fusion Technology **21** (1992) 1557

[60] N. P. Taylor, R. A. Forrest, P. J. Knight and L. J. Baker, *"Safety and Environmental Modelling in the PROCESS Code"*, Strategic Studies Note 94/14 (1994)

[61] R. L. Reid and Y-K. M. Peng, *"Potential Minimum Cost of Electricity of Superconducting Coil Tokamak Power Reactors"*, Proceedings of 13th IEEE Symposium on Fusion Engineering, Knoxville, Tennessee, October 1989, p. 258

[62] J. Sheffield et al., *"Cost Assessment of a Generic Magnetic Fusion Reactor"*, *Fusion Technology* **9** (1986) 199

[63] S. Thompson, *"Systems Code Cost Accounting"*, memo FEDC-M-88-SE,-004 (1988)

[64] J. D. Galambos, L. J. Perkins, S. W. Haney and J. Mandrekas, *Nuclear Fusion*, **35** (1995) 551

[65] J. P. Holdren et al., *"Report of the Senior Committee on Environmental Safety and Economic Aspects of Magnetic Fusion Energy"*, *Fusion Technology*, **13** (1988) 7

[66] Git version control system `http://git-scm.com/`

[67] K. Schittkowski, *"Nonlinear Programming Codes"*, Springer, Berlin, 1980

[68] D. G. Luenberger and Yinyu Ye, *"Linear and Nonlinear Programming"*, International Series in Operations Research and Management Science, 3rd Edition, Springer Science and Business Media LCC, 2008

[69] M. J. D. Powell, *"An efficient method for finding the minimum of a function of several variables without calculating derivatives"*, *Computer Journal*, **7** No. 2 (1964) 155–162

[70] S.-P. Han, *"A Globally Convergent Method for Nonlinear Programming"*, Department for Computer Science, Cornell University, Report 75-257 (1975)

[71] M. S. Bazaraa, H. D. Sherali and C. M. Shetty, *"Nonlinear Programming: Theory and Algorithms"*, John Wiley & Sons, Inc., New York, 1993

[72] R. Fletcher, *"A General Quadratic Programming Algorithm"*, Atomic Energy Research Establishment, Report T.P. 401, March 1970

[73] R. Fletcher, *"The calculation of feasible points for linearly constrained optimisation problems"*, Atomic Energy Research Establishment, Report R.6354, April 1970

[74] R. Fletcher, *"A FORTRAN subroutine for general quadratic programming"*, Atomic Energy Research Establishment, Report R.6370, June 1970

[75] M. J. D. Powell, *"The convergence of variable metric methods for nonlinearly constrained optimisation calculations"*, presented at Nonlinear Programming Symposium 3, Madison, Wisconsin, 1977

Appendix A

The Optimisation Solver Explained

To give the user a better understanding of the optimisation solver implemented in PROCESS and the interpretation of its results, we give a short introduction into the mathematical background for solving these type of problems as well as the specific algorithm used.

In section A.1, we the general nonlinear programming problem is defined which is the mathematical description of our problem. This problem is typically formulated using Lagrange multipliers (c.f. A.2) and is solved numerically most efficiently using sequential quadratic programming (c.f. A.3). The Fortran subroutine that is used to implement such an optimisation solver in PROCESS is called VMCON (c.f. A.4), which iterates between solving a local quadratic subproblem (c.f. A.5) and a one dimensional line search (c.f. A.6). As the total method is using a quasi-Newtonian approach the Hessian matrix is approximated by a variant of the Broyden-Fletcher-Goldfarb-Shanno update (A.7). Section A.8 summarises the symbol convention used in this chapter.

A.1 The General Nonlinear Programming Problem

Mathematically the *general nonlinear programming problem* or *nonlinear constrained optimisation problem* is defined as

$$\text{minimise } f(\boldsymbol{x}), \tag{A.1a}$$

$$\text{subject to } c_i(\boldsymbol{x}) = 0, \qquad i = 1, \ldots, k, \tag{A.1b}$$

$$\text{and } c_i(\boldsymbol{x}) \geq 0, \qquad i = k+1, \ldots, m, \tag{A.1c}$$

where both the *objective function*[1] $f(\boldsymbol{x})$ and the *constraints* $c_i(\boldsymbol{x})$ are nonlinear functions of the n-dimensional vector of variables $\boldsymbol{x}$ with bounds $\boldsymbol{x} \in \Omega$. In this context, all $\boldsymbol{x} \in \Omega$ that fulfill the constraints $c_i(\boldsymbol{x})$ are called *feasible*. They describe the allowed space in which we are trying to optimise the objective function $f(\boldsymbol{x})$. Note that any maximisation problem can be written as a minimisation by using $f'(\boldsymbol{x}) = -f(\boldsymbol{x})$ and that any equality constraint $c(\boldsymbol{x}) = a$ can be rewritten as $c'(\boldsymbol{x}) = c(\boldsymbol{x}) - a = 0$. Any inequality constraint can therefore be rearranged analogously to fit the form described in eq. A.1c.

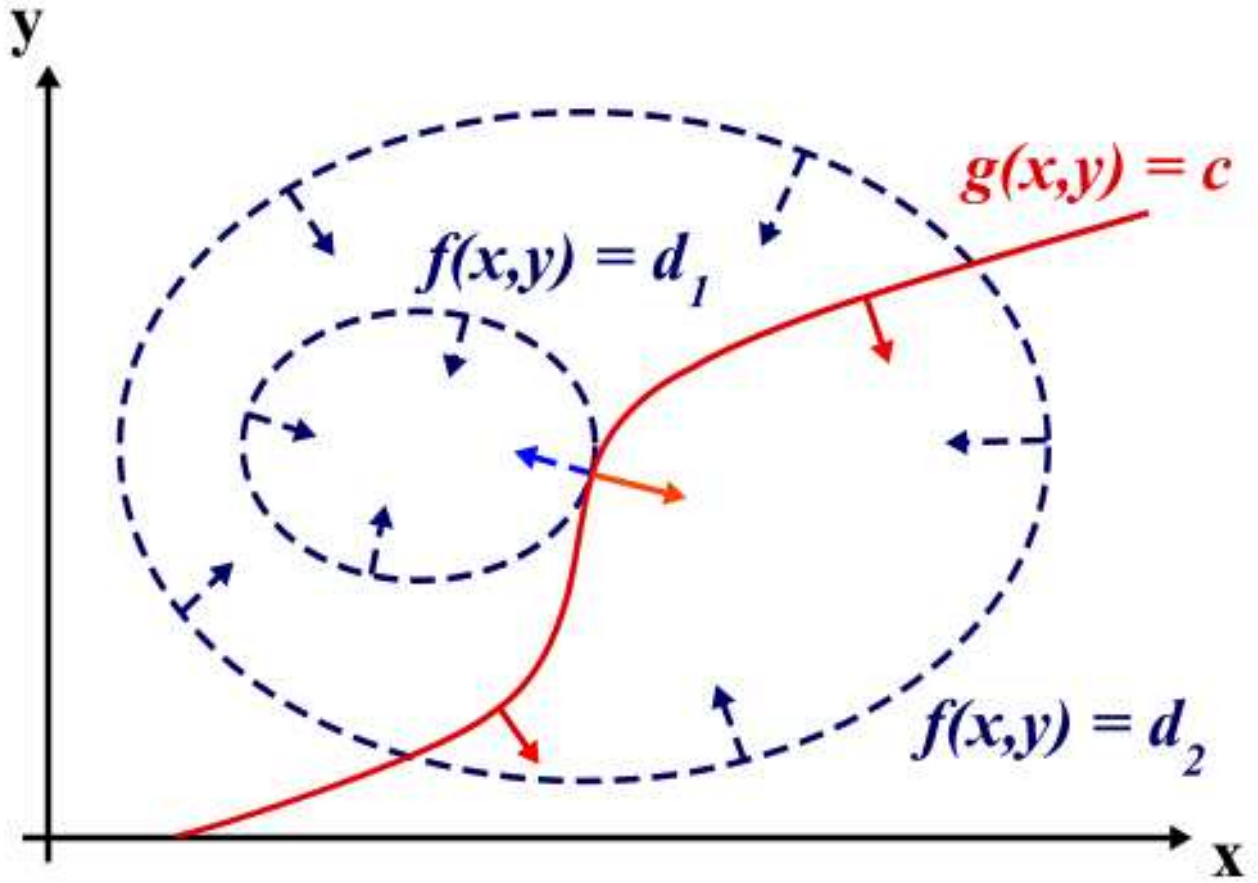

Figure A.1: *Illustration of Lagrange Multiplier Method (credit Wikipedia) showing two contour lines of the objective function $f(x, y) = d_i$ (dark blue dashed lines) and the nonlinear constraint $g(x, y) = c$ (red solid line) as well as their gradients (blue and red arrows) at various positions including the constrained optimum (light blue and orange arrows).*

A.2 The Lagrange Method

The general nonlinear programming problem can be solved mathematically using Lagrange's method. It assumes that the constraints cannot be used to explicitly reduce the parameter space of the iteration variables - as it is typically the case for non-linear constraints and objective functions - and is therefore a powerful method applicable to a general class of problems.

If we assume for simplicity that we have a 2D problem with only one equality constraint $c(x, y) = 0$, we know that we only need to search for the optimum along that constraint. At the optimum, the value of the objective function will then be stationary, i.e. it does not locally increase or decrease along the constraint. As the gradient of a function is perpedicular to its contour lines of $f(x, y) = d$, this is equivalent to saying that the gradient of the objective function at that point is parallel to the gradient of the constraints

$$\nabla f(x, y) = -\lambda \nabla c(x, y), \tag{A.2}$$

where the factor λ is necessary as only the direction, but not the magnitude nor the sign of the gradients need to be equal. This is also illustrated in Figure A.1.

When expanding the method to several equality and inequality constraints we can make use of the *Lagrange function*. For the nonlinear programming problem described by A.1 it is given by

$$L(\boldsymbol{x}, \boldsymbol{\lambda}) = f(\boldsymbol{x}) - \sum_{i=1}^{m} \lambda_i c_i(\boldsymbol{x}), \tag{A.3}$$

with the corresponding *Lagrangian multipliers* λ_i. It allows us to formulate the *first order necessary* conditions for a constrained optimum $\boldsymbol{x}^*$ with corresponding Lagrange multipliers $\boldsymbol{\lambda}^*$, the Karush-Kuhn-Tucker (KKT) conditions,

$$\nabla_x L(\boldsymbol{x}^*, \boldsymbol{\lambda}^*) = \nabla_x f(\boldsymbol{x}^*) - \sum_{i=1}^{m} \lambda_i \nabla_x c_i(\boldsymbol{x}^*) = 0, \tag{A.4a}$$

$$\lambda_i^* c_i(\boldsymbol{x}^*) = 0, \qquad i = 1, \ldots, m, \tag{A.4b}$$

$$c_i(\boldsymbol{x}^*) = 0, \qquad i = 1, \ldots, k, \tag{A.4c}$$

$$\lambda_i^* \geq 0, \qquad i = k+1, \ldots, m, \tag{A.4d}$$

$$c_i(\boldsymbol{x}^*) \geq 0, \qquad i = k+1, \ldots, m. \tag{A.4e}$$

Please note that by construction the Lagrange multipliers λ_i^* fulfilling the KKT conditions are describing the derivative of the objective function with respect to the constraint equation $\frac{df}{dc_i}$ and are therefore a measure of how much the objective function changes with respect to each constraint.

In the special case of a *continuously differentiable convex* objective function $f(\boldsymbol{x})$ and equality constraints as well as *affine* inequality constraints, these KKT conditions are also sufficient for a global optimum. The `PROCESS` optimisation solver has been designed to converge on the KKT conditions, but does not test whether these are *sufficient* for a global optimum. It is therefore crucial that the user verifies that a global optimum has been found.

Furthermore, these conditions and therefore the solver, assume that both the objective function and constraints are at least *first order continuously partially differential*, i.e. that their first order partial derivatives are all continuous functions. This might not always be the case in `PROCESS` and is a potential source of errors.

[1] Please note that what is called *figure of merit* in `PROCESS` is called *objective function* in the mathematical optimisation context. Hence, both names are used equivalently in this document.

A.3 Sequential Quadratic Programming (SQP)

Based on the Lagrange method, sequential (also successive or recursive) quadratic programming is the most efficient method to numerically solve constrained nonlinear optimisation problems [67]. It combines a simple solver for a quadratic optimisation problem with linear constraints that determines a search direction $\boldsymbol{\delta}$ with a line search to find an optimum along that search direction. It is a type of the more general *feasible direction* methods which is itself a type of *primal method* that solve nonlinear programming problems in the $n - m$ dimensional feasible subspace [68].

A.4 VMCON

The optimisation solver implemented in `PROCESS` is the Fortran routine `VMCON` [5]. It is a modified version of the `vf02ad` routine from the Harwell Software Library[2] and implements a SQP method originally suggested by Powell[3] [6] based on work by Han [70]. As stated before `VMCON` is designed to converge on a solution of the *necessary* KKT conditions A.4, but does not check the *sufficient* conditions. Its convergence criterion is therefore given by

$$\left|\nabla_x f(\boldsymbol{x}^{j-1})^T \cdot \boldsymbol{\delta}^j\right| + \sum_{i=1}^{m} \left|\lambda_i^j c_i(\boldsymbol{x}^{j-1})\right| < \texttt{epsvmc} \tag{A.5}$$

where `epsvmc` is a user specified error tolerance, $\boldsymbol{\delta}^j$ is the vector in direction of the next line search (c.f. section A.6) and j is the counter of the sequential quadratic programming iteration. Hence, the first part estimates the predicted improvement due to the next line search and the second part measures the error in the complimentary condition A.4b of the KKT conditions.

Figure A.2 describes the flow chart of the `VMCON` routine, while the various values of the return parameter `ifail`[4] are described and interpreted in Table A.1.

A.5 The Quadratic Subproblem (QSP)

Within sequential quadratic programming the complex nonlinear problem is broken down into solving a sequence of local quadratic subproblems with linear constraints. This is based on the assumption that locally the problem is well approximated by a second order Taylor expansion of the Lagrange function. Hence, the local quadratic subproblem is described by

$$\text{minimise } Q(\boldsymbol{\delta}) = f(\boldsymbol{x}^{j-1}) + \boldsymbol{\delta}^T \nabla_x f(\boldsymbol{x}^{j-1}) + \frac{1}{2}\boldsymbol{\delta}^T \nabla_{xx} L(\boldsymbol{x}^{j-1}, \boldsymbol{\lambda}^{j-1})\boldsymbol{\delta} \tag{A.6a}$$

$$\text{subject to } \boldsymbol{\delta}^T \nabla_x c_i(\boldsymbol{x}^{j-1}) + c_i(\boldsymbol{x}^{j-1}) = 0,\ i = 1, \ldots, k, \tag{A.6b}$$

$$\text{and } \boldsymbol{\delta}^T \nabla_x c_i(\boldsymbol{x}^{j-1}) + c_i(\boldsymbol{x}^{j-1}) \geq 0,\ i = k+1, \ldots, m, \tag{A.6c}$$

where $\boldsymbol{\delta} = \boldsymbol{x} - \boldsymbol{x}^{j-1}$, the index $j-1$ indicates the parameter values of the previous iteration[5] and $\nabla_{xx} L(\boldsymbol{x}^{j-1}, \boldsymbol{\lambda}^{j-1})$ is the Hessian of the Lagrange function. The solution of the QSP $\boldsymbol{\delta}$ describes the change of the current iteration variable vector that minimises the local approximation of the problem. To assure convergence even from bad starting points, it is not directly used to update the iteration

[2] www.hsl.rl.ac.uk

[3] This should not be confused with Powell's algorithm [69] which solves a multidimensional unconstrained minimisation problem without derivatives.

[4] Note, within the `VMCON` routine `ifail` is called `info`.

[5] Note j is called `nqp` in the `VMCON` routine.

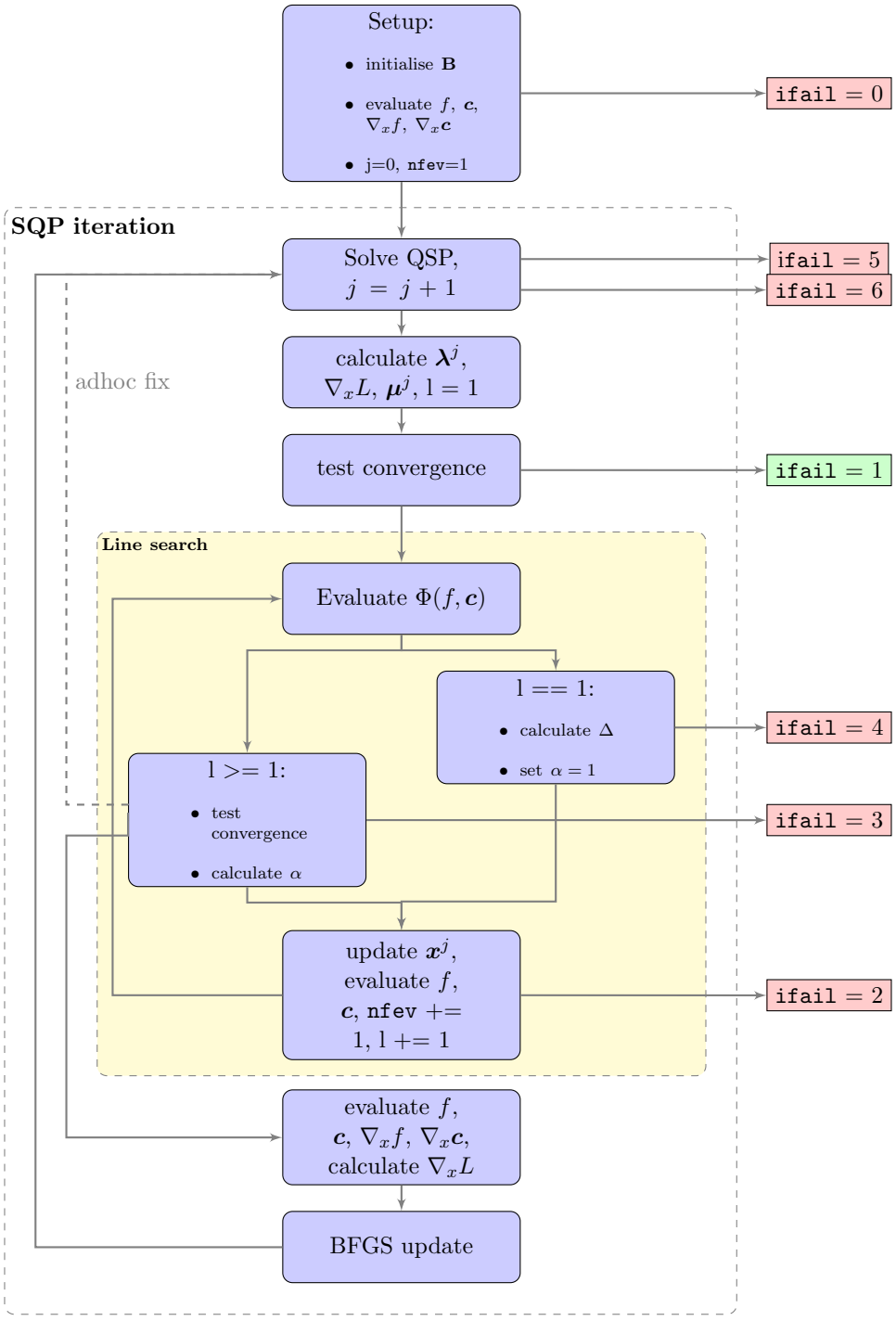

Figure A.2: *This is the flow chart of the* `VMCON` *optimisation solver. The criteria for and the interpretation of the successful (*`ifail` $= 1$*) and unsuccessful (*`ifail` $\neq 1$*) return parameters are described in Table A.1.*

`ifail`	Description	Meaning	Recommendation
0	`VMCON`: Improper input parameters	The input parameters to the solver are wrong, e.g. negative number of iteration variables.	This needs to be fixed on the developer side and should only occur in the test phase of new modules.
1	`VMCON`: Normal return	`VMCON` has found a solution that fulfills the necessary conditions within the specified error tolerances (c.f. eq. A.5).	Test whether the solution is a global solution by using different starting parameters.
2	Too many function calls	During the line search `VMCON` has reached the maximum number of total function calls (`maxfev`=100)	`VMCON` struggles to find a solution. Retry with different start parameters.
3	Line search required 10 functions calls	The results produced by input objective function/constraints and their gradients are inconsistent. This can be due to numerical noise in the input functions.	The developer needs to check the consistency and numerical robustness of the objective function, constraints and their derivatives. Perhaps the accurracy in numerical integrations/differentiations needs to be higher. As a user, try changing the iteration bounds or adding other iteration variables.
4	Uphill search direction was calculated	The quadratic subproblem has suggested a search direction in which the objective function only increases.	This happens if an inconsistency between the objective function, the constraints and their respective derivatives occurs (c.f. `ifail` =3).
5	`qpsub`: No feasible solution or bad approximation of Hessian	Either no optimum lies within the space allowed by the constraints and variable bounds or the identity matrix is not a good first approximation of the Hessian.	As a user, add a new iteration variable, as developer, try using a multiple of the identity matrix as initial Hessian instead.
6	`qpsub`: Singular matrix in quadratic subproblem or restriction by artificial bounds	This is fairly self-explanatory.	If this is meaningful, widen the boundaries of the iteration variables.
7	Line search has been aborted	ADD SOME STUFF HERE (CHECK WITH MDK)	

Table A.1: Summary of the description and meaning of the `VMCON` return parameters `ifail`.

variable vector, but describes the direction of the line search in which a 1d function is minimised using a Newtonian method (c.f. section A.6). Being a second order approximation to the original problem, it typically has a faster convergence that sequential linear programming (SLP) methods [71, chap. 10.4].

To allow the applicability of the solver to more general problems, Powell [6] substituted the Hessian $\nabla_{xx}L(\boldsymbol{x}^{j-1}, \boldsymbol{\lambda}^{j-1})$ with a positive definite approximation $\mathbf{B}$. This means that both the objective function $f(\boldsymbol{x})$ and the constraint equations $c_i(\boldsymbol{x})$ only have to be continuously differentiable instead of twice continuously differentiable with respect to the iteration variable vector $\boldsymbol{x}$. This makes the method a Quasi-Newtonian method as opposed to true Newtonian methods. How this approximation is initialised and updated is described in more detail in the section about the Broyden-Fletcher-Goldfarb-Shanno update A.7.

To solve the QSP `VMCON` uses `harwqp` a modification of the the Harwell library routine `VE02AD`, which in itself uses the subroutine `harwfp/LA02AD` to find a feasible point within the variable bounds and linearised constraints. Both routines go back to a method by Fletcher [72, 73, 74]. The Lagrange multipliers are also determined from results of the `harwqp` routine.

If the routine cannot find a feasible point it fails with `ifail` = 5 (c.f. Table A.1). As the routine is only checking the local linear approximation of the constraints rather the full non-linear constraints, there is a chance that a feasible point exists even though the routine fails with `ifail` = 5. In these cases, it is possible that the first approximation of the Hessian has not been good and the algorithm has, therefore, taken an inappropriately large step. Then using a multiple of the identity matrix will improve convergence of the algorithm.

If a singular matrix is encountered with the QSP solver or the solution is restricted by the artificial bounds, `VMCON` fails with `ifail` = 6. In this case it can be helpful to widen the boundaries of the iteration variables.

A.6 The Line Search

The line search is an essential part of the SQP algorithm. As Powell [6] pointed out, it is necessary to allow convergence from poor starting conditions. It uses the vector $\boldsymbol{\delta}$ from the QSP to update $\boldsymbol{x}^j = \boldsymbol{x}^{j-1} + \alpha^j \boldsymbol{\delta}^j$ where the step-length parameter $\alpha^j > 0$ is determined by the line search as the parameter that minimises

$$\Phi(\alpha) = f(\boldsymbol{x}) + \sum_{i=1}^{k} \mu_i \left| c_i(\boldsymbol{x}) \right| + \sum_{i=k+1}^{m} \mu_i \left| min(0, c_i(\boldsymbol{x})) \right| \tag{A.7}$$

where the weights are defined as

$$\mu_i = \begin{cases} |\lambda_i^1| & \text{if } j = 1, \\ max\left(|\lambda_i^j|, 1/2(\mu_i^{j-1} + |\lambda_i^j|) \right) & \text{if } j > 1 \end{cases} \tag{A.8}$$

to assure maximally efficient convergence [70]. Note, that in this method locally *inactive* inequality constraints (c.f. 2.4.2) are not having any effect. It should always be possible, to find a solution that fulfills

$$\Phi(\alpha = 0) > \Phi(\alpha^j), \tag{A.9}$$

if

$$\left. \frac{d\Phi}{d\alpha} \right|_{\alpha=0} < 0. \tag{A.10}$$

In case the derivative is positive (`dflsa` ≥ 0), an uphill search direction has been determined and the code stops with `ifail` $= 4$. This typically only happens, if the objective function or constraints are inconsistent with their own derivatives.

As the line search tries to determine the optimum of a one dimensional, but fully non-linear function $\Phi(\alpha)$, it creates a series of α_l values[6]. At each iteration l, a quadratic local function $\Phi_l(\alpha)$ fullfilling the boundary conditions $\Phi_l(0) = \Phi(0)$, $\Phi_l'(0) = \Delta$ and $\Phi_l(\alpha_{l-1}) = \Phi(\alpha_{l-1})$ is minimised, where typically $\Delta = \Phi'(0)$. This leads to

$$\Phi_l(\alpha) = \Phi(0) + \Delta\alpha + \frac{\Phi(\alpha_{l-1}) - \Phi(0) - \Delta\alpha_{l-1}}{\alpha_{l-1}^2}\alpha^2 \tag{A.11}$$

and minimising this gives

$$\alpha_{min} = -\frac{\Delta\alpha_{l-1}^2}{2(\Phi(\alpha_{l-1}) - \Phi(0) - \Delta\alpha_{l-1})}. \tag{A.12}$$

Powell then sets $\alpha_l = \min(\alpha_{min}, 0.1\alpha_{l-1})$. On each iteration the convergence of the line search is tested. It is reached, if

$$\Phi(\alpha_l) - \Phi(0) < 0.1\Delta. \tag{A.13}$$

which assures that the change in the function is small in comparison to its derivative and 0.1 is a factor determined by experience. If this criterion is successful, the code exits the line search and updates all function evaluations.

As a modification to the original code, we have added an adhoc fix that exits the line search in the case of

$$\Phi(\alpha_l) - \Phi(0) > 0. \tag{A.14}$$

This has been added as experience have shown that `VMCON` typically does not converge in these situations, but if it is forced to caculate a new search direction in this way, it sometimes successfully finishes. Note, that this cannot force `VMCON` to converge on any false solutions, as it only exits positively when the convergence criterion A.5 is fulfilled.

Typically, the line search already converges after one iteration and, therefore, $\alpha = 1$. Hence, the `VMCON` line search has an upper limit of maximally 10 iterations before it terminating with `ifail` $= 3$ (c.f. Table A.1). This is higher than Powell's original limit of 5 to avoid a few cases of early termination without a major effect on efficiency.

Within the line search `VMCON` also checks that the total number of function calls has not exceeded `maxfev` $= 100$. If this is the case, it exits with error code `ifail` $= 2$. Both checks assure that the routine stops, if it does not seem to be able to find a solution.

A.7 The Broyden-Fletcher-Goldfarb-Shanno (BFGS) quasi-Newton update

`VMCON` uses a quasi-Newtonian update, the Broyden-Fletcher-Goldfarb-Shanno update, in the approximation of the Hessian of the Lagrange function. This means it is applicable to all continously differentialbe objective and constraint functions. Note, that due to the constraints it is not actually necessary for the Hessian of the solution to be positive definite, even though this is essential for the convergence of the algorithm.

In quasi-Newtonian methods the identity matrix $\mathbf{I}$ is often chosen as an initial estimate of the Hessian because of its positive definiteness. In some cases it can be helpful to chose a constant multiple of the

[6] In the actual code $\alpha =$`calpha` and $\alpha_l =$`alpha`$*\alpha_{l-1}$.

identity matrix instead. The BGFS update [7] is one way of revising the initial Hessian approximation using the update of the iteration variable vector

$$\boldsymbol{\xi} = \boldsymbol{x}^j - \boldsymbol{x}^{j-1} \tag{A.15}$$

and the update of the Jacobian of the Lagrange function

$$\boldsymbol{\gamma} = \nabla_x L(\boldsymbol{x}^j, \boldsymbol{\lambda}^j) - \nabla_x L(\boldsymbol{x}^{j-1}, \boldsymbol{\lambda}^j). \tag{A.16}$$

Note that unless $\alpha = 1$ in the line search, $\boldsymbol{\xi} \neq \boldsymbol{\delta}$. To assure the positive definiteness of $\mathbf{B}$ Powell [6] suggested a variation of the standard formalism that uses $\boldsymbol{\eta}$ instead of $\boldsymbol{\gamma}$

$$\boldsymbol{\eta} = \theta\boldsymbol{\gamma} + (1-\theta)\mathbf{B}\boldsymbol{\xi} \tag{A.17}$$

with

$$\theta = \begin{cases} 1 & \boldsymbol{\xi}^T\boldsymbol{\gamma} \geq 0.2\boldsymbol{\xi}^T\mathbf{B}\boldsymbol{\xi} \\ \frac{0.8\boldsymbol{\xi}^T B\boldsymbol{\xi}}{\boldsymbol{\xi}^T} & \boldsymbol{\xi}^T\boldsymbol{\gamma} < 0.2\boldsymbol{\xi}^T\mathbf{B}\boldsymbol{\xi} \end{cases} \tag{A.18}$$

to calculate a new approximation of the Hessian

$$\mathbf{B}_{new} = \mathbf{B} - \frac{\mathbf{B}\boldsymbol{\xi}\boldsymbol{\xi}^T\mathbf{B}}{\boldsymbol{\xi}^T\mathbf{B}\boldsymbol{\xi}} + \frac{\boldsymbol{\eta}^T\boldsymbol{\eta}}{\boldsymbol{\xi}^T\boldsymbol{\eta}}. \tag{A.19}$$

Using the modified version of the BFGS update as suggested by Powell, furthermore, assures superlinear convergence, even if the true Hessian is indefinite [75].

A.8 Symbols

In the previous sections the following conventions have been assumed: $\boldsymbol{x}$, $\boldsymbol{\delta}$, $\boldsymbol{\xi}$, $\boldsymbol{\gamma}$ and $\boldsymbol{\eta}$ are n-dimensional vectors, where n is the number of iteration variables. $\boldsymbol{c}$, $\boldsymbol{\lambda}$, $\boldsymbol{\mu}$ are m-dimensional vectors, where m is the total number of constraints and c_i, λ_i and μ_i $(i = 1 \ldots m)$ are their respective components. $\mathbf{B}$ and $\mathbf{I}$ are $n \times n$-dimensional matrices, while $\nabla_x \boldsymbol{c}$ is an $n \times m$-dimensional matrix.

Appendix B

Non-optimisation Input File

The following is a typical input file used to run PROCESS in non-optimisation mode. Comments in [...] have been added to the right of each line.

```
* Numerics information                [Comment]

NEQNS = 14,                           [Number of active constraint equations]
NVAR = 14,                            [Number of active iteration variables]
ICC =   1,  2, 10, 11, 7, 16, 24, 5, 31, 32, 33, 34, 35, 36,  [Constraint eqns]
ixc =   5, 10, 12, 3,  7,  6, 36, 9, 48, 49, 50, 51, 53, 54,  [Iteration variables]
IOPTIMZ = -1,                         [Turn off optimisation]
ISWEEP=0                              [No scans (non-optimisation mode)]

* F-values and limits

FBETATRY = 1.0                        [N.B. active iteration variable 36]

* Physics parameters

ASPECT = 3.5,                         [Machine aspect ratio]
BETA = 0.042,                         [N.B. active iteration variable 5]
BT = 6.,                              [Toroidal field on axis]
DENE = 1.5e20,                        [N.B. active iteration variable 6]
FVSBRNNI = 1.0,                       [Non-inductive volt seconds fraction]
DNBETA = 3.5,                         [Beta g coefficient]
HFACT = 2.,                           [N.B. active iteration variable 10]
ICURR = 4,                            [Use ITER current scaling]
ISC = 6,                              [Use ITER 89-P confinement time scaling law]
IINVQD = 1,                           [Use inverse quadrature]
IITER = 1,                            [Use ITER fusion power calculations]
ISHAPE = 0,                           [Use input values for KAPPA and TRIANG]
KAPPA = 2.218,                        [Plasma elongation]
Q = 3.0,                              [Edge safety factor]
RMAJOR = 7.0,                         [N.B. active iteration variable 3]
RNBEAM = 0.0002,                      [N.B. active iteration variable 7]
TBURN = 227.9                         [Burn time]
TE = 15.,                             [Electron temperature]
TRIANG = 0.6                          [Plasma triangularity]

* Current drive parameters

IRFCD = 1,                            [Use current drive]
IEFRF = 5                             [Use ITER neutral beam current drive]
FEFFCD = 3.,                          [Artificially enhance efficiency]
```

```
* Divertor parameters

ANGINC=0.262,                          [Angle of incidence of field lines on plate]
PRN1=0.285                             [Density ratio]

* Machine build

BORE = 0.12,                           [Machine bore]
OHCTH = 0.1,                           [central solenoid thickness]
GAPOH = 0.08,                          [Inboard gap]
TFCTH = 0.9,                           [Inboard TF coil leg thickness]
DDWI = 0.07,                           [Vacuum vessel thickness]
SHLDITH = 0.69,                        [Inboard shield thickness]
BLNKITH = 0.115,                       [Inboard blanket thickness]
FWITH = 0.035,                         [Inboard first wall thickness]
SCRAPLI = 0.14,                        [Inboard scrape-off layer thickness]
SCRAPLO = 0.15,                        [Outboard scrape-off layer thickness]
FWOTH = 0.035,                         [Outboard first wall thickness]
BLNKOTH = 0.235,                       [Outboard blanket thickness]
SHLDOTH = 1.05,                        [Outboard shield thickness]
GAPOMIN = 0.21,                        [Outboard gap]
VGAPTF = 0,                            [Vertical gap]

* First wall, blanket, shield parameters

LBLNKT=0                               [Use old blanket model]
DENSTL=7800.                           [Steel density]

* TF coil parameters

OACDCP = 1.4e7,                        [N.B. active iteration variable 12]
ITFSUP = 1,                            [Use superconducting TF coils]
RIPMAX = 5.,                           [Maximum TF ripple]

* PF coil parameters

NGRP = 3,                              [Three groups of PF coils]
IPFLOC = 1,2,3,                        [Locations for each group]
NCLS = 2,2,2,1,                        [Number of coils in each group]
COHEOF = 1.85e7,                       [central solenoid current at End Of Flat-top]
FCOHBOP = 0.9,                         [central solenoid current at Begin. Of Pulse / COHEOF]
ROUTR = 1.5,                           [Radial position for group 3]
ZREF(3) = 2.5,                         [Z position for group 3]
OHHGHF = .71                           [Height ratio central solenoid / TF coil]

* Vacuum system parameters

NTYPE = 1                              [Use cryopump]

* Heat transport parameters

ETATH=0.35                             [Thermal to electric conversion efficiency]
FMGDMW = 0.                            [Power to MGF units]
BASEEL=5.e6                            [Base plant electric load]
ISCENR=2                               [Energy store option]

* Buildings

FNDT = 2.                              [Foundation thickness]
EFLOOR=1.d5                            [Effective total floor space]
```

```
* Costs

IREACTOR = 1,                          [Calculate cost of electricity]
IFUELTYP = 0                           [Treat blanket, first wall etc as capital cost]
UCHRS = 87.9,                          [Unit cost of heat rejection system]
UCCPCL1 = 250,                         [Unit cost of high strength tapered copper]
UCCPCLB = 150                          [Unit cost of TF outer leg plate coils]
```

Appendix C

Optimisation Input File

The following is a typical input file used to run PROCESS in optimisation mode. Comments in [...] have been added to the right of each line.

```
* Numerics information                [Comment]

boundl(1) = 2.5,                     [Bound on iteration variable 1 (aspect)]
BOUNDU(10) = 3.,                     [Bound on iteration variable 10 (hfact)]
BOUNDU(60) = 4.d4,                   [Bound on iteration variable 60 (cpttf)]
NEQNS = 15,                          [Number of active constraint equations]
NVAR = 25                            [Number of active iteration variables]
ICC =   1,  2, 10, 11, 7, 16, 8, 24, 31, 32, 33, 34, 35, 36, 14, [Constraint eqns]
ixc =   5, 10, 12, 3,  7,        36, 48, 49, 50, 51, 53, 54, 19, [Corresponding...]
        1, 2, 4, 6, 13, 16, 29, 56, 57, 58, 59, 60,              [...iteration variables]
IOPTIMZ = 1,                         [Turn on optimisation]
MINMAX = 6,                          [Minimise cost of electricity]

ISWEEP=7,                            [Seven point scan]
NSWEEP=6,                            [Use WALALW as scanning variable]
SWEEP= 6.0,5.5,4.5,4.0,3.5,3.0,2.5   [Values of WALALW for each scan point]

* F-values and limits

FBETATRY = 1.0                       [N.B. active iteration variable 36]

* Physics parameters

ASPECT = 3.5,                        [N.B. active iteration variable 1]
BETA = 0.042,                        [N.B. active iteration variable 5]
BT = 6.,                             [N.B. active iteration variable 2]
DENE = 1.5e20,                       [N.B. active iteration variable 6]
FVSBRNNI = 1.0,                      [Non-inductive volt seconds fraction]
DNBETA = 3.5,                        [Beta g coefficient]
HFACT = 2.,                          [N.B. active iteration variable 10]
ICURR = 4,                           [Use ITER current scaling]
ISC = 6,                             [Use ITER 89-P confinement time scaling law]
IINVQD = 1,                          [Use inverse quadrature]
IITER = 1,                           [Use ITER fusion power calculations]
ISHAPE = 0,                          [Use input values for KAPPA and TRIANG]
KAPPA = 2.218,                       [Plasma elongation]
Q = 3.0,                             [Edge safety factor]
RMAJOR = 7.0,                        [N.B. active iteration variable 3]
RNBEAM = 0.0002,                     [N.B. active iteration variable 7]
TBURN = 227.9                        [Burn time]
```

```
TE = 15.,                           [N.B. active iteration variable 4]
TRIANG = 0.6                        [Plasma triangularity]

* Current drive parameters

IRFCD = 1,                          [Use current drive]
IEFRF = 5                           [Use ITER neutral beam current drive]
FEFFCD = 3.,                        [Artificially enhance efficiency]

* Divertor parameters

ANGINC=0.262,                       [Angle of incidence of field lines on plate]
PRN1=0.285                          [Density ratio]

* Machine build

BORE = 0.12,                        [N.B. active iteration variable 29]
OHCTH = 0.1,                        [N.B. active iteration variable 16]
GAPOH = 0.08,                       [Inboard gap]
TFCTH = 0.9,                        [N.B. active iteration variable 13]
DDWI = 0.07,                        [Vacuum vessel thickness]
SHLDITH = 0.69,                     [Inboard shield thickness]
BLNKITH = 0.115,                    [Inboard blanket thickness]
FWITH = 0.035,                      [Inboard first wall thickness]
SCRAPLI = 0.14,                     [Inboard scrape-off layer thickness]
SCRAPLO = 0.15,                     [Outboard scrape-off layer thickness]
FWOTH = 0.035,                      [Outboard first wall thickness]
BLNKOTH = 0.235,                    [Outboard blanket thickness]
SHLDOTH = 1.05,                     [Outboard shield thickness]
GAPOMIN = 0.21,                     [Outboard gap]
VGAPTF = 0,                         [Vertical gap]

* First wall, blanket, shield parameters

LBLNKT=0                            [Use old blanket model]
DENSTL=7800.                        [Steel density]

* TF coil parameters

OACDCP = 1.4e7,                     [N.B. active iteration variable 12]
ITFSUP = 1,                         [Use superconducting TF coils]
RIPMAX = 5.,                        [Maximum TF ripple]

* PF coil parameters

NGRP = 3,                           [Three groups of PF coils]
IPFLOC = 1,2,3,                     [Locations for each group]
NCLS = 2,2,2,1,                     [Number of coils in each group]
COHEOF = 1.85e7,                    [central solenoid current at End Of Flat-top]
FCOHBOP = 0.9,                      [central solenoid current at Begin. Of Pulse / COHEOF]
ROUTR = 1.5,                        [Radial position for group 3]
ZREF(3) = 2.5,                      [Z position for group 3]
OHHGHF = .71                        [Height ratio central solenoid / TF coil]

* Vacuum system parameters

NTYPE = 1                           [Use cryopump]

* Heat transport parameters

ETATH=0.35                          [Thermal to electric conversion efficiency]
FMGDMW = 0.                         [Power to MGF units]
```

```
BASEEL=5.e6                          [Base plant electric load]
ISCENR=2                             [Energy store option]

* Buildings

FNDT = 2.                            [Foundation thickness]
EFLOOR=1.d5                          [Effective total floor space]

* Costs

IREACTOR = 1,                        [Calculate cost of electricity]
IFUELTYP = 0                         [Treat blanket, first wall etc as capital cost]
UCHRS = 87.9,                        [Unit cost of heat rejection system]
UCCPCL1 = 250,                       [Unit cost of high strength tapered copper]
UCCPCLB = 150                        [Unit cost of TF outer leg plate coils]
```

Appendix D

Source Code Documentation

The development of the PROCESS code since its shipment from Oak Ridge National Laboratory in April 1992 has been fully documented in the Project Work File [31]. Presented here is a list of Project Work File Notes as of September 30, 2014 that address various issues related to the source code.

- Documentation of each individual source routine is an ongoing task. A Work File Note will be produced as each routine is processed, with the eventual aim of bringing all these together into a single document, i.e. a future edition of this manual.
- Summary of work performed since April 1992 : Note 0160
- Code status (routine SCCS version numbers) : Note 0165
- SCCS (Source Code Control System) implementation for PROCESS: Note 0003
- Present code standard : Note 0160
- Future code standard (to be adhered to) : Note 0167
- Directory structure and location of all relevant files : Note 0168
- Proposed future work : Note 0160
- Pulsed power plant coding : Note 0189
- Nuclide inventory and activation (FISPACT) module : Notes 0195 and 0231
- New cost algorithms : Note 0224
- Stellarator model : Note 0246
- D-^{3}He tokamak model : Note 0265

Reference [31] has recently been superseded by a new Project Work File, F/MI/PJK/PROCESS:-

- Spherical Tokamak model: Note 001
- Reversed Field Pinch model: Note 009
- Test cases: Note 019